Tanzu Application Platform v1.5

VMware Tanzu Application Platform 1.5

vmware by Broadcom

You can find the most up-to-date technical documentation on the VMware by Broadcom website at:

https://docs.vmware.com/

VMware by Broadcom
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

Get started with Learning Center 1071

Installing Learning Center 1071

ClusterRunTemplate reference for Supply Chain Choreographer 1512

Tanzu Application Platform v1.5

VMware Tanzu Application Platform (commonly known as TAP) is an application development platform with a rich set of developer tools. It offers developers a paved path to production to build and deploy software quickly and securely on any compliant public cloud or on-premises Kubernetes cluster.

Tanzu Application Platform overview

Tanzu Application Platform:

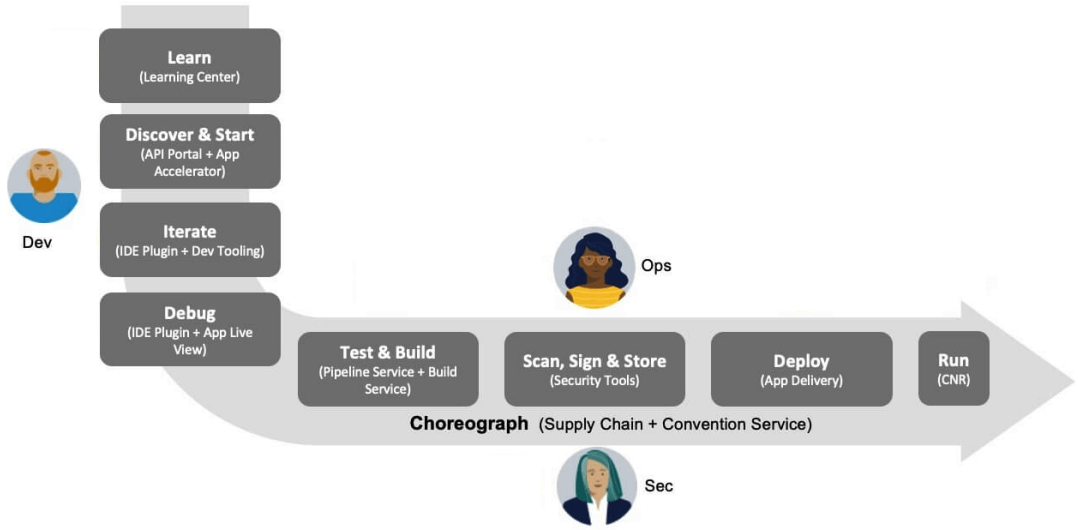
- Delivers a superior developer experience for enterprises building and deploying cloud-native applications on Kubernetes.
- Allows developers to quickly build and test applications regardless of their familiarity with Kubernetes.
- Helps application teams get to production faster by automating source-to-production pipelines.
- Clearly defines the roles of developers and operators so they can work collaboratively and integrate their efforts.

Operations teams can create application scaffolding templates with built-in security and compliance guardrails, making those considerations mostly invisible to developers. Starting with the templates, developers turn source code into a container and get a URL to test their app in minutes.

After the container is built, it updates every time there's a new code commit or dependency patch. An internal API management portal facilitates connecting to other applications and data, regardless of how they're built or the infrastructure they run on.

Simplified workflows

When creating supply chains, you can simplify workflows in both the inner and outer loop of Kubernetes-based app development with Tanzu Application Platform.



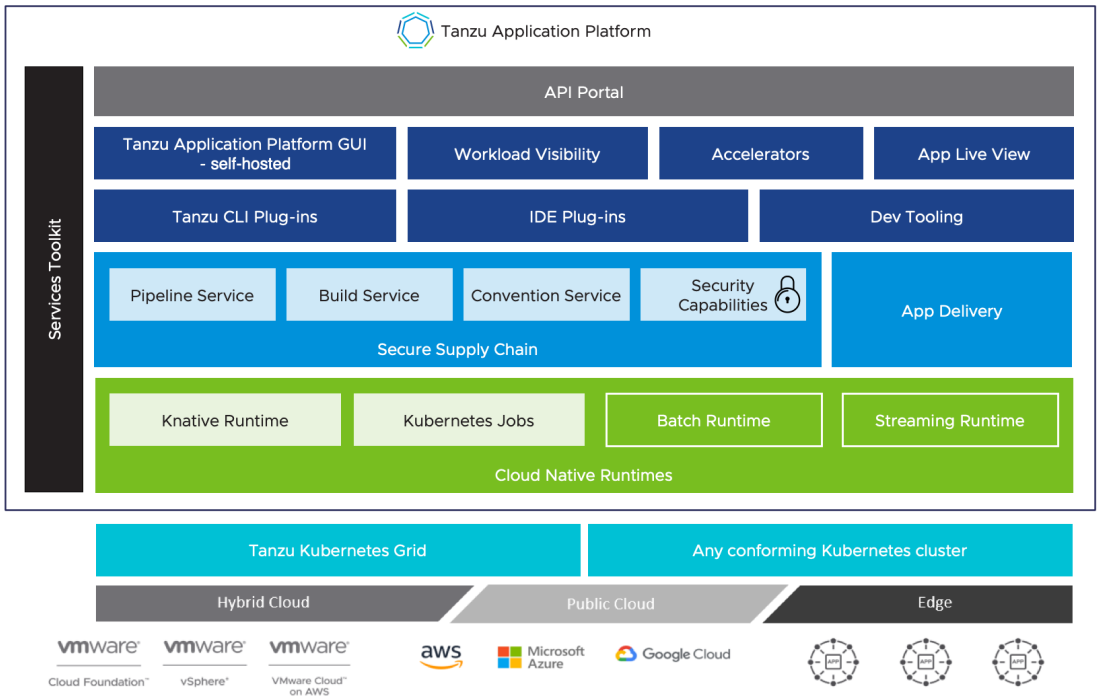
- **Inner Loop**
    - The inner loop describes a developer's development cycle of iterating on code.
    - Inner loop activities include coding, testing, and debugging before making a commit.
    - On cloud-native or Kubernetes platforms, developers in the inner loop often build container images and connect their apps to all necessary services and APIs to deploy them to a development environment.
- **Outer Loop**
    - The outer loop describes how operators deploy apps to production and maintain them over time.
    - On a cloud-native platform, outer loop activities include:
        - Building container images.
        - Adding container security.
        - Configuring continuous integration and continuous delivery (CI/CD) pipelines.
    - Outer loop activities are challenging in a Kubernetes-based development environment. App delivery platforms are constructed from various third-party and open source components with numerous configuration options.
- **Supply Chains and choreography**
    - Tanzu Application Platform uses the choreography pattern inherited from the context of microservices^1 and applies it to CI/CD to create a path to production.^2

Supply chains provide a way of codifying all of the steps of your path to production, or what is more commonly known as CI/CD. A supply chain differs from CI/CD in that with a supply chain, you can add every step necessary for an application to reach production or a lower environment.

To address the developer experience gap, the path to production allows users to create a unified access point for all of the tools required for their applications to reach a customer-facing environment.

Instead of having separate tools that are loosely coupled to each other for testing and building, security, deploying, and running apps, a path to production defines all four tools in a single, unified layer of abstraction. Where tools typically can't integrate with one another and additional scripting or webhooks are necessary, a unified automation tool codifies all interactions between each of the tools.

Tanzu Application Platform provides a default set of components that automates pushing an app to staging and production on Kubernetes. This removes the pain points for both inner and outer loops. It also allows operators to customize the platform by replacing Tanzu Application Platform components with other products.



For more information about Tanzu Application Platform components, see Components and installation profiles.

Notice of telemetry collection for Tanzu Application Platform

Tanzu Application Platform participates in the VMware Customer Experience Improvement Program (CEIP). As part of CEIP, VMware collects technical information about your organization's use of VMware products and services in association with your organization's VMware license keys. For information about CEIP, see the Trust & Assurance Center. You may join or leave CEIP at any time. The CEIP Standard Participation Level provides VMware with information to improve its products and services, identify and fix problems, and advise you on how to best deploy and use VMware products. For example, this information can enable a proactive product deployment discussion with your VMware account team or VMware support team to help resolve your issues. This information cannot directly identify any individual.

You must acknowledge that you have read the VMware CEIP policy before you can proceed with the installation. For more information, see Install your Tanzu Application Platform profile. To opt out of telemetry participation after installation, see Opting out of telemetry collection.

Tanzu Application Platform release notes

This topic describes the changes in Tanzu Application Platform (commonly known as TAP) v1.5.

v1.5.12

Release Date: 09 April 2024

v1.5.12 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
metadata-store.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-rcjv-mgp8-qvmr • GHSA-qppj-fm5r-hxr3 • GHSA-m425-mq94-257g • GHSA-4374-p667-p6c8 • GHSA-2wrh-6pvc-2jm9 • CVE-2023-45285 • CVE-2023-44487 • CVE-2023-39325 • CVE-2023-39323
sso.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2024-26462 • CVE-2024-26461 • CVE-2024-26458 • CVE-2024-0553 • CVE-2023-7008 • CVE-2023-5981 • CVE-2023-4813 • CVE-2023-4806 • CVE-2022-48303 • CVE-2021-36087 • CVE-2021-36086 • CVE-2021-36085 • CVE-2021-36084

Package Name	Vulnerabilities Resolved
tap-gui.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-30589 • CVE-2023-30588 • CVE-2023-2650 • CVE-2023-0842 • CVE-2023-0466 • CVE-2023-0465 • CVE-2022-4415 • CVE-2022-3821 • CVE-2020-36634 • CVE-2020-17753 • CVE-2019-9705 • CVE-2019-9704 • CVE-2017-9525 • CVE-2013-1779 • CVE-2006-1611 • CVE-2002-1647

v1.5.12 Known issues

This release introduces no new known issues.

v1.5.11

Release Date: 12 March 2024

v1.5.11 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
spring-cloud-gateway.tanzu.vmware.com	▼ Expand to see the list • GHSA-hr8g-6v94-x4m9 • GHSA-ccgv-vj62-xf9h • GHSA-4g9r-vxhx-9pgx • GHSA-45x7-px36-x8w8 • GHSA-4265-ccf5-phj5 • CVE-2024-26308 • CVE-2024-25710 • CVE-2024-22365 • CVE-2024-20952 • CVE-2024-20932 • CVE-2024-20926 • CVE-2024-20918 • CVE-2024-0727 • CVE-2024-0567 • CVE-2024-0553 • CVE-2023-6237 • CVE-2023-6129 • CVE-2023-5678 • CVE-2023-4641 • CVE-2023-39326

Package Name	Vulnerabilities Resolved
tekton.tanzu.vmware.com	▼ Expand to see the list • GHSA-hp87-p4gw-j4gq • CVE-2024-22365 • CVE-2024-0567 • CVE-2024-0553 • CVE-2023-7192 • CVE-2023-6918 • CVE-2023-6546 • CVE-2023-6004 • CVE-2023-5981 • CVE-2023-5717 • CVE-2023-5363 • CVE-2023-5197 • CVE-2023-5178 • CVE-2023-5158 • CVE-2023-5156 • CVE-2023-4921 • CVE-2023-4881 • CVE-2023-48795 • CVE-2023-4813 • CVE-2023-4806 • CVE-2023-47038 • CVE-2023-4623 • CVE-2023-4622 • CVE-2023-46218 • CVE-2023-45871 • CVE-2023-45863 • CVE-2023-45862 • CVE-2023-4569 • CVE-2023-44487 • CVE-2023-44466 • CVE-2023-42756 • CVE-2023-42755 • CVE-2023-42754 • CVE-2023-42753 • CVE-2023-42752 • CVE-2023-4273 • CVE-2023-4244 • CVE-2023-4208 • CVE-2023-4207 • CVE-2023-4206 • CVE-2023-4194 • CVE-2023-4155

Package Name	Vulnerabilities Resolved
	• CVE-2023-4147 • CVE-2023-4132 • CVE-2023-4128 • CVE-2023-40283 • CVE-2023-4016 • CVE-2023-4015 • CVE-2023-4004 • CVE-2023-3995 • CVE-2023-39804 • CVE-2023-39198 • CVE-2023-39197 • CVE-2023-39194 • CVE-2023-39193 • CVE-2023-39192 • CVE-2023-39189 • CVE-2023-3866 • CVE-2023-3865 • CVE-2023-3863 • CVE-2023-38546 • CVE-2023-38432 • CVE-2023-38429 • CVE-2023-38428 • CVE-2023-38426 • CVE-2023-3817 • CVE-2023-3777 • CVE-2023-3776 • CVE-2023-3773 • CVE-2023-3772 • CVE-2023-37453 • CVE-2023-3611 • CVE-2023-3610 • CVE-2023-3609 • CVE-2023-36054 • CVE-2023-35829 • CVE-2023-35828 • CVE-2023-35824 • CVE-2023-35823 • CVE-2023-35788 • CVE-2023-3567 • CVE-2023-35001 • CVE-2023-3446 • CVE-2023-3439 • CVE-2023-34319

Package Name	Vulnerabilities Resolved
	• CVE-2023-34256 • CVE-2023-3390 • CVE-2023-3389 • CVE-2023-3358 • CVE-2023-3357 • CVE-2023-3355 • CVE-2023-3338 • CVE-2023-33288 • CVE-2023-33203 • CVE-2023-3268 • CVE-2023-32269 • CVE-2023-32248 • CVE-2023-32233 • CVE-2023-3220 • CVE-2023-3212 • CVE-2023-3161 • CVE-2023-31484 • CVE-2023-31436 • CVE-2023-3141 • CVE-2023-31248 • CVE-2023-3117 • CVE-2023-31085 • CVE-2023-31084 • CVE-2023-31083 • CVE-2023-3090 • CVE-2023-30772 • CVE-2023-30456 • CVE-2023-2985 • CVE-2023-2975 • CVE-2023-29491 • CVE-2023-29007 • CVE-2023-2898 • CVE-2023-28466 • CVE-2023-28328 • CVE-2023-28322 • CVE-2023-28321 • CVE-2023-27538 • CVE-2023-27536 • CVE-2023-27535 • CVE-2023-27534 • CVE-2023-27533 • CVE-2023-26607 • CVE-2023-26606

Package Name	Vulnerabilities Resolved
	• CVE-2023-26605 • CVE-2023-26545 • CVE-2023-26544 • CVE-2023-2650 • CVE-2023-2612 • CVE-2023-2603 • CVE-2023-2602 • CVE-2023-25815 • CVE-2023-25775 • CVE-2023-25652 • CVE-2023-25588 • CVE-2023-25585 • CVE-2023-25584 • CVE-2023-25012 • CVE-2023-24329 • CVE-2023-23946 • CVE-2023-23916 • CVE-2023-23915 • CVE-2023-23914 • CVE-2023-23559 • CVE-2023-23455 • CVE-2023-23454 • CVE-2023-23004 • CVE-2023-2283 • CVE-2023-2269 • CVE-2023-22490 • CVE-2023-2235 • CVE-2023-2194 • CVE-2023-2166 • CVE-2023-2163 • CVE-2023-2162 • CVE-2023-2156 • CVE-2023-21400 • CVE-2023-21255 • CVE-2023-2124 • CVE-2023-21102 • CVE-2023-20938 • CVE-2023-20593 • CVE-2023-20588 • CVE-2023-20569 • CVE-2023-2006 • CVE-2023-2002 • CVE-2023-1998

Package Name	Vulnerabilities Resolved
	• CVE-2023-1990 • CVE-2023-1972 • CVE-2023-1872 • CVE-2023-1859 • CVE-2023-1855 • CVE-2023-1829 • CVE-2023-1670 • CVE-2023-1667 • CVE-2023-1652 • CVE-2023-1611 • CVE-2023-1513 • CVE-2023-1382 • CVE-2023-1380 • CVE-2023-1281 • CVE-2023-1255 • CVE-2023-1206 • CVE-2023-1195 • CVE-2023-1192 • CVE-2023-1079 • CVE-2023-1078 • CVE-2023-1077 • CVE-2023-1076 • CVE-2023-1075 • CVE-2023-1074 • CVE-2023-1073 • CVE-2023-0597 • CVE-2023-0468 • CVE-2023-0465 • CVE-2023-0464 • CVE-2023-0461 • CVE-2023-0459 • CVE-2023-0458 • CVE-2023-0394 • CVE-2023-0386 • CVE-2023-0361 • CVE-2023-0266 • CVE-2023-0210 • CVE-2023-0179 • CVE-2023-0045 • CVE-2022-48522 • CVE-2022-48502 • CVE-2022-4842 • CVE-2022-48425

Package Name	Vulnerabilities Resolved
	• CVE-2022-48424 • CVE-2022-48423 • CVE-2022-48303 • CVE-2022-47929 • CVE-2022-47696 • CVE-2022-47673 • CVE-2022-47521 • CVE-2022-47520 • CVE-2022-47519 • CVE-2022-47518 • CVE-2022-47011 • CVE-2022-47010 • CVE-2022-47008 • CVE-2022-47007 • CVE-2022-45919 • CVE-2022-45886 • CVE-2022-45869 • CVE-2022-45703 • CVE-2022-44840 • CVE-2022-4415 • CVE-2022-4382 • CVE-2022-4379 • CVE-2022-4285 • CVE-2022-4269 • CVE-2022-42329 • CVE-2022-42328 • CVE-2022-4139 • CVE-2022-4129 • CVE-2022-41218 • CVE-2022-40982 • CVE-2022-3996 • CVE-2022-3821 • CVE-2022-3707 • CVE-2022-36280 • CVE-2022-3545 • CVE-2022-3521 • CVE-2022-35205 • CVE-2022-3435 • CVE-2022-3424 • CVE-2022-3344 • CVE-2022-3169 • CVE-2022-27672 • CVE-2022-2196

Package Name	Vulnerabilities Resolved
	• CVE-2007-4559

v1.5.11 Known issues

This release introduces no new known issues.

v1.5.10

Release Date: 13 February 2024

v1.5.10 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
application-configuration-service.tanzu.vmware.com	▼ Expand to see the list • GHSA-wjxj-5m7g-mg7q • GHSA-cgwf-w82q-5jrr • GHSA-45x7-px36-x8w8 • CVE-2023-42503
carbonblack.scanning.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-6xv5-86q9-7xr8 • GHSA-6wrf-mxfj-pf5p • GHSA-33pg-m6jh-5237

Package Name	Vulnerabilities Resolved
ootb-templates.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • GHSA-p782-xgp4-8hr8 • GHSA-mw99-9chc-xw7r • GHSA-9763-4f94-gfch • GHSA-7ww5-4wqc-m92c • GHSA-449p-3h89-pw88 • GHSA-2wrh-6pvc-2jm9 • CVE-2024-22365 • CVE-2024-0567 • CVE-2024-0553 • CVE-2024-0193 • CVE-2023-7192 • CVE-2023-6932 • CVE-2023-6931 • CVE-2023-6918 • CVE-2023-6817 • CVE-2023-6606 • CVE-2023-6546 • CVE-2023-6040 • CVE-2023-6004 • CVE-2023-5981 • CVE-2023-5717 • CVE-2023-5363 • CVE-2023-5197 • CVE-2023-5178 • CVE-2023-5158 • CVE-2023-5156 • CVE-2023-4921 • CVE-2023-4911 • CVE-2023-4881 • CVE-2023-48795 • CVE-2023-4813 • CVE-2023-4806 • CVE-2023-47038 • CVE-2023-4623 • CVE-2023-4622 • CVE-2023-46218 • CVE-2023-45871 • CVE-2023-45862 • CVE-2023-4569 • CVE-2023-44466 • CVE-2023-42756

Package Name	Vulnerabilities Resolved
	• CVE-2023-42755 • CVE-2023-42754 • CVE-2023-42753 • CVE-2023-42752 • CVE-2023-4273 • CVE-2023-4244 • CVE-2023-4208 • CVE-2023-4207 • CVE-2023-4206 • CVE-2023-4194 • CVE-2023-4155 • CVE-2023-4147 • CVE-2023-4132 • CVE-2023-4128 • CVE-2023-40283 • CVE-2023-4016 • CVE-2023-4015 • CVE-2023-4004 • CVE-2023-3995 • CVE-2023-39804 • CVE-2023-39198 • CVE-2023-39197 • CVE-2023-39194 • CVE-2023-39193 • CVE-2023-39192 • CVE-2023-39189 • CVE-2023-3866 • CVE-2023-3865 • CVE-2023-3863 • CVE-2023-38546 • CVE-2023-38545 • CVE-2023-38432 • CVE-2023-38429 • CVE-2023-38428 • CVE-2023-38426 • CVE-2023-3817 • CVE-2023-3777 • CVE-2023-3776 • CVE-2023-3773 • CVE-2023-3772 • CVE-2023-37453 • CVE-2023-3611 • CVE-2023-3610

Package Name	Vulnerabilities Resolved
	• CVE-2023-3609 • CVE-2023-36054 • CVE-2023-35829 • CVE-2023-35828 • CVE-2023-35824 • CVE-2023-35823 • CVE-2023-35788 • CVE-2023-35001 • CVE-2023-3446 • CVE-2023-3439 • CVE-2023-34319 • CVE-2023-34256 • CVE-2023-3390 • CVE-2023-3389 • CVE-2023-3338 • CVE-2023-33288 • CVE-2023-33203 • CVE-2023-3268 • CVE-2023-32248 • CVE-2023-3212 • CVE-2023-3141 • CVE-2023-31248 • CVE-2023-3117 • CVE-2023-31085 • CVE-2023-31084 • CVE-2023-31083 • CVE-2023-3090 • CVE-2023-30772 • CVE-2023-2975 • CVE-2023-2898 • CVE-2023-28466 • CVE-2023-28322 • CVE-2023-28321 • CVE-2023-25775 • CVE-2023-23004 • CVE-2023-2269 • CVE-2023-2235 • CVE-2023-2194 • CVE-2023-2163 • CVE-2023-2156 • CVE-2023-21400 • CVE-2023-21255 • CVE-2023-2124

Package Name	Vulnerabilities Resolved
	• CVE-2023-20593 • CVE-2023-20588 • CVE-2023-20569 • CVE-2023-2002 • CVE-2023-1990 • CVE-2023-1855 • CVE-2023-1611 • CVE-2023-1206 • CVE-2023-1192 • CVE-2023-0597 • CVE-2022-48522 • CVE-2022-48502 • CVE-2022-48425 • CVE-2022-47011 • CVE-2022-47010 • CVE-2022-47008 • CVE-2022-47007 • CVE-2022-45919 • CVE-2022-45886 • CVE-2022-45703 • CVE-2022-44840 • CVE-2022-4285 • CVE-2022-4269 • CVE-2022-40982 • CVE-2022-35205
sso.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-45145 • CVE-2023-41056 • CVE-2023-41053 • CVE-2023-39319 • CVE-2023-39318 • CVE-2023-3817 • CVE-2023-36054 • CVE-2023-3446 • CVE-2023-29409 • CVE-2023-29406

v1.5.10 Resolved issues

The following issues, listed by component and area, are resolved in this release.

v1.5.10 Resolved issues: Application Single Sign-On

- When requesting an `access_token` by using the the Authorization Code flow, scopes in the token are filtered based on user roles. In this version, the `scope` parameter of the access token response is also filtered, with the same rules. For more information, see the OAuth documentation.

v1.5.10 Resolved issues: Contour

- Ships with Contour v1.24.6.
- Supports upgrades to Tanzu Application Platform v1.5.10 without downtime when transitioning from `DaemonSet` to `Deployments`.

> **Note**
>
> Downtime-free upgrades require more than one nodes in the cluster.

v1.5.10 Known issues

This release introduces no new known issues.

v1.5.9

Release Date: 09 January 2024

v1.5.9 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
api-portal.tanzu.vmware.com	▼ Expand to see the list • GHSA-vmq6-5m68-f53m • CVE-2023-5981 • CVE-2023-47038 • CVE-2023-4016 • CVE-2023-36054 • CVE-2022-48522
application-configuration-service.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-6378 • CVE-2023-44487 • CVE-2023-39325 • CVE-2023-3635 • CVE-2023-34053

Package Name	Vulnerabilities Resolved
cnrs.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-5363 • CVE-2023-5156 • CVE-2023-4813 • CVE-2023-4806 • CVE-2023-45285 • CVE-2023-39326 • CVE-2023-3817 • CVE-2023-3446 • CVE-2023-2975 • CVE-2023-2650 • CVE-2023-1255 • CVE-2023-0465 • CVE-2023-0464 • CVE-2022-3996
metadata-store.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2022-41717 • CVE-2022-41715 • CVE-2022-2880 • CVE-2022-2879
spring-cloud-gateway.tanzu.vmware.com	▼ Expand to see the list • GHSA-jjfh-589g-3hjx • CVE-2023-5981 • CVE-2023-5156 • CVE-2023-4813 • CVE-2023-4806 • CVE-2023-47038 • CVE-2023-4016 • CVE-2023-39804 • CVE-2023-34053 • CVE-2022-48522

Package Name	Vulnerabilities Resolved
sso.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-vmq6-5m68-f53m • CVE-2023-5363 • CVE-2023-5156 • CVE-2023-34053 • CVE-2023-34035 • CVE-2023-2975 • CVE-2023-22049 • CVE-2023-22045 • CVE-2023-22044 • CVE-2023-22041 • CVE-2023-22036 • CVE-2023-22006 • CVE-2023-20863 • CVE-2023-20861

v1.5.9 Known issues

This release has the following known issues, listed by component and area.

v1.5.9 Known issues: Supply Chain Security Tools - Scan

- The Snyk scanner outputs an incorrectly created date, resulting in an invalid date. If the workload is in a failed state due to an invalid date, wait approximately 10 hours and the workload automatically goes into the ready state. For more information, see this issue in the Snyk Github repository.

v1.5.8

Release Date: 12 December 2023

v1.5.8 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
api-portal.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • CVE-2023-5363 • CVE-2023-3817 • CVE-2023-3446 • CVE-2023-2975 • CVE-2023-22081 • CVE-2023-22025

Package Name	Vulnerabilities Resolved
apis.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • GHSA-2wrh-6pvc-2jm9 • CVE-2023-5363 • CVE-2023-3817 • CVE-2023-3446 • CVE-2023-2975

Package Name	Vulnerabilities Resolved
buildservice.tanzu.vmware.com	▼ Expand to see the list • GHSA-hp87-p4gw-j4gq • GHSA-4374-p667-p6c8 • CVE-2023-5981 • CVE-2023-5363 • CVE-2023-5197 • CVE-2023-4921 • CVE-2023-4911 • CVE-2023-4881 • CVE-2023-4623 • CVE-2023-4622 • CVE-2023-45871 • CVE-2023-44487 • CVE-2023-44466 • CVE-2023-42756 • CVE-2023-42755 • CVE-2023-42753 • CVE-2023-42752 • CVE-2023-4273 • CVE-2023-4244 • CVE-2023-4208 • CVE-2023-4207 • CVE-2023-4206 • CVE-2023-4194 • CVE-2023-4155 • CVE-2023-4132 • CVE-2023-4016 • CVE-2023-3866 • CVE-2023-3865 • CVE-2023-3863 • CVE-2023-38546 • CVE-2023-38545 • CVE-2023-38432 • CVE-2023-38429 • CVE-2023-38428 • CVE-2023-38426 • CVE-2023-3817 • CVE-2023-3772 • CVE-2023-36054 • CVE-2023-35829 • CVE-2023-35828 • CVE-2023-35824 • CVE-2023-35823

Package Name	Vulnerabilities Resolved
	• CVE-2023-3446 • CVE-2023-34319 • CVE-2023-34256 • CVE-2023-3338 • CVE-2023-3268 • CVE-2023-32248 • CVE-2023-3212 • CVE-2023-31484 • CVE-2023-3141 • CVE-2023-31085 • CVE-2023-31084 • CVE-2023-31083 • CVE-2023-2975 • CVE-2023-29491 • CVE-2023-2898 • CVE-2023-2650 • CVE-2023-2603 • CVE-2023-2602 • CVE-2023-25775 • CVE-2023-23004 • CVE-2023-2269 • CVE-2023-2235 • CVE-2023-2163 • CVE-2023-2156 • CVE-2023-21255 • CVE-2023-2124 • CVE-2023-1255 • CVE-2023-1206 • CVE-2023-1192 • CVE-2023-0465 • CVE-2023-0464 • CVE-2022-48502 • CVE-2022-48425 • CVE-2022-40982 • CVE-2022-3996

Package Name	Vulnerabilities Resolved
cnrs.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-39319 • CVE-2023-39318 • CVE-2023-29409 • CVE-2023-29406 • CVE-2023-29403 • CVE-2023-24536 • CVE-2023-24534 • CVE-2023-24532
developer-conventions.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • GHSA-4374-p667-p6c8 • GHSA-2wrh-6pvc-2jm9 • CVE-2023-5363 • CVE-2023-3817 • CVE-2023-3446 • CVE-2023-2975 • CVE-2023-2650 • CVE-2023-1255 • CVE-2023-0465 • CVE-2023-0464 • CVE-2022-3996

Package Name	Vulnerabilities Resolved
eventing.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • GHSA-m425-mq94-257g • GHSA-4374-p667-p6c8 • GHSA-2wrh-6pvc-2jm9 • CVE-2023-44487 • CVE-2023-39323 • CVE-2023-39319 • CVE-2023-39318 • CVE-2023-29409 • CVE-2023-29406 • CVE-2023-29405 • CVE-2023-29404 • CVE-2023-29403 • CVE-2023-29402 • CVE-2023-29400 • CVE-2023-24540 • CVE-2023-24539 • CVE-2023-24538 • CVE-2023-24537 • CVE-2023-24536 • CVE-2023-24534 • CVE-2023-24532 • CVE-2022-41725 • CVE-2022-41724 • CVE-2022-41723 • CVE-2022-41722 • CVE-2022-41717 • CVE-2022-41715 • CVE-2022-32189 • CVE-2022-32148 • CVE-2022-30635 • CVE-2022-30633 • CVE-2022-30632 • CVE-2022-30631 • CVE-2022-30630 • CVE-2022-2880 • CVE-2022-2879 • CVE-2022-28131 • CVE-2022-27664 • CVE-2022-1962 • CVE-2022-1705

Package Name	Vulnerabilities Resolved
spring-cloud-gateway.tanzu.vmware.com	▼ Expand to see the list • GHSA-xpw8-rcwv-8f8p • GHSA-vmq6-5m68-f53m • GHSA-qppj-fm5r-hxr3 • GHSA-jgvc-jfgh-rjvv • CVE-2023-5363 • CVE-2023-4911 • CVE-2023-44487 • CVE-2023-3817 • CVE-2023-36054 • CVE-2023-3446 • CVE-2023-2975 • CVE-2023-22081 • CVE-2023-22025

v1.5.8 Known issues

This release introduces no new known issues.

v1.5.7

Release Date: 14 November 2023

v1.5.7 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
api-portal.tanzu.vmware.com	▼ Expand to see the list • GHSA-2wrh-6pvc-2jm9 • CVE-2023-2603 • CVE-2023-2602 • CVE-2023-22049 • CVE-2023-22045 • CVE-2023-22044 • CVE-2023-22041 • CVE-2023-22036 • CVE-2023-22006
contour.tanzu.vmware.com	▼ Expand to see the list • GHSA-qppj-fm5r-hxr3 • GHSA-m425-mq94-257g • GHSA-4374-p667-p6c8 • GHSA-2wrh-6pvc-2jm9 • CVE-2023-44487

v1.5.7 Known issues

This release has the following known issues, listed by component and area.

v1.5.7 Known issues: Tanzu Application Platform

- Tanzu Application Platform v1.5.7 is not supported with Tanzu Kubernetes releases (TKR) v1.26 on vSphere with Tanzu v8.

---

v1.5.6

Release Date: 10 October 2023

v1.5.6 Breaking changes

This release has the following breaking changes, listed by component and area.

v1.5.6 Breaking changes: Services Toolkit

- Services Toolkit forces explicit cluster-wide permissions to `claim` from a `ClusterInstanceClass`. You must now grant the permission to `claim` from a `ClusterInstanceClass` by using a `ClusterRole` and `ClusterRoleBinding`. For more information, see The claim verb for ClusterInstanceClass.

---

v1.5.6 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
accelerator.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-43642
api-portal.tanzu.vmware.com	▼ Expand to see the list • GHSA-6mjq-h674-j845 • CVE-2023-31484 • CVE-2023-29491 • CVE-2023-2650 • CVE-2023-1255

Package Name	Vulnerabilities Resolved
apis.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-31484 • CVE-2023-29491 • CVE-2023-29383 • CVE-2023-26604 • CVE-2023-2650 • CVE-2023-0465 • CVE-2023-0464 • CVE-2022-3821 • CVE-2022-3219 • CVE-2020-13844 • CVE-2016-2781 • CVE-2013-4235
apiserver.appliveview.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-2650 • CVE-2023-1255 • CVE-2023-0465 • CVE-2023-0464 • CVE-2022-3996
application-configuration-service.tanzu.vmware.com	▼ Expand to see the list • GHSA-mjmq-gwgm-5qhm • GHSA-3p86-9955-h393 • CVE-2023-20863 • CVE-2023-20861
buildservice.tanzu.vmware.com	▼ Expand to see the list • CVE-2022-48064
controller.source.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-6wrf-mxfj-pf5p • GHSA-33pg-m6jh-5237
conventions.appliveview.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-2650 • CVE-2023-1255 • CVE-2023-0465 • CVE-2023-0464 • CVE-2022-3996
learningcenter.tanzu.vmware.com	▼ Expand to see the list • CVE-2022-48064 • CVE-2022-45919 • CVE-2022-45887 • CVE-2021-3712

Package Name	Vulnerabilities Resolved
ootb-templates.tanzu.vmware.com	▼ Expand to see the list • CVE-2022-48064 • CVE-2022-45919 • CVE-2022-45887
policy.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-2650 • CVE-2023-1255 • CVE-2023-0465 • CVE-2023-0464 • CVE-2022-3996
services-toolkit.tanzu.vmware.com	▼ Expand to see the list • GHSA-hp87-p4gw-j4gq
spring-cloud-gateway.tanzu.vmware.com	▼ Expand to see the list • GHSA-cgwf-w82q-5jrr • GHSA-7g45-4rm6-3mm3 • GHSA-5mg8-w23w-74h3 • CVE-2023-42503 • CVE-2023-3635 • CVE-2023-2976 • CVE-2023-22049 • CVE-2023-22045 • CVE-2023-22044 • CVE-2023-22041 • CVE-2023-22036 • CVE-2023-22006 • CVE-2020-8908
tap-gui.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-32559 • CVE-2023-32006
tekton.tanzu.vmware.com	▼ Expand to see the list • CVE-2022-48566 • CVE-2022-48565 • CVE-2022-48564 • CVE-2022-48560 • CVE-2022-48064 • CVE-2022-45919 • CVE-2022-45887

Package Name	Vulnerabilities Resolved
workshops.learningcenter.tanzu.vmware.com	▼ Expand to see the list • CVE-2022-48064 • CVE-2022-45919 • CVE-2022-45887 • CVE-2021-3712

v1.5.6 Resolved issues

The following issues, listed by component and area, are resolved in this release.

v1.5.6 Resolved issues: Application Configuration Service

- Resolves an issue which caused client applications that include the `spring-cloud-config-client` dependency to fail to start or properly load the configuration that Application Configuration Service produced. The fix is adding the property `spring.cloud.config.enabled=false` in secret resources that Application Configuration Service produced.
- Resolves some installation failure scenarios by setting the pod security context to adhere to the restricted pod security standard.

v1.5.6 Known issues

This release has the following known issues, listed by component and area.

v1.5.6 Known issues: Tanzu Application Platform

- Tanzu Application Platform v1.5.6 is not supported with Tanzu Kubernetes releases (TKR) v1.26 on vSphere with Tanzu v8.

v1.5.5

Release Date: 12 September 2023

v1.5.5 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
buildservice.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-35788 • CVE-2023-3439 • CVE-2023-32233 • CVE-2023-3220 • CVE-2023-31436 • CVE-2023-3117 • CVE-2023-30456 • CVE-2023-2985 • CVE-2023-2612 • CVE-2023-25012 • CVE-2023-2283 • CVE-2023-1667 • CVE-2023-1380 • CVE-2022-4415 • CVE-2022-3821
carbonblack.scanning.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-2q89-485c-9j2x
eventing.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-2650 • CVE-2023-1255 • CVE-2023-0465 • CVE-2023-0464 • CVE-2022-3996
learningcenter.tanzu.vmware.com	▼ Expand to see the list • GHSA-cf7p-gm2m-833m • CVE-2023-4147 • CVE-2023-4015 • CVE-2023-4004 • CVE-2023-3995 • CVE-2023-3777 • CVE-2023-3635 • CVE-2023-3610 • CVE-2023-3609 • CVE-2022-45886
ootb-templates.tanzu.vmware.com	▼ Expand to see the list • CVE-2022-45886
spring-cloud-gateway.tanzu.vmware.com	▼ Expand to see the list • CVE-2022-41881

Package Name	Vulnerabilities Resolved
tap-gui.tanzu.vmware.com	▼ Expand to see the list • GHSA-cchq-frgv-rjh5 • GHSA-g644-9gfx-q4q4
tekton.tanzu.vmware.com	▼ Expand to see the list • CVE-2022-45886
workshops.learningcenter.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-4147 • CVE-2023-4015 • CVE-2023-4004 • CVE-2023-3995 • CVE-2023-3777 • CVE-2023-3635 • CVE-2023-3610 • CVE-2023-3609 • CVE-2022-45886

v1.5.5 Resolved issues

The following issues, listed by component and area, are resolved in this release.

v1.5.5 Resolved issues: Application Configuration Service

- `GitRepository` is now consistently observed beyond 15 minutes. The `interval` property for a `ConfigurationSlice` now continues to work as expected.
- Error-logging is improved where a `ConfigurationSlice` references a non-existent `ConfigurationSource`. A `ConfigurationSlice` properly reconciles after the referenced `ConfigurationSource` is created.

v1.5.5 Resolved issues: Tanzu CLI and plugins

- This release includes Tanzu CLI v1.2.0 and a set of installable plug-in groups that are versioned so that the CLI is compatible with every supported version of Tanzu Applicatin Platform. For more information, see Install Tanzu CLI.

v1.5.5 Known issues

This release has the following known issues, listed by component and area.

v1.5.5 Known issues: Tanzu Application Platform

- Tanzu Application Platform v1.5.5 is not supported with Tanzu Kubernetes releases (TKR) v1.26 on vSphere with Tanzu v8.

v1.5.4

Release Date: 15 August 2023

v1.5.4 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
carbonblack.scanning.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-g2j6-57v7-gm8c • GHSA-m8cg-xc2p-r3fc
controller.source.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-hw7c-3rfg-p46j • CVE-2023-2650 • CVE-2023-1255

ootb-templates.tanzu.vmware.com	▼ Expand to see the list • GHSA-m8cg-xc2p-r3fc • GHSA-hw7c-3rfg-p46j • GHSA-g2j6-57v7-gm8c • CVE-2023-3567 • CVE-2023-3358 • CVE-2023-3357 • CVE-2023-32269 • CVE-2023-32233 • CVE-2023-3220 • CVE-2023-3161 • CVE-2023-31484 • CVE-2023-31436 • CVE-2023-30456 • CVE-2023-2985 • CVE-2023-29491 • CVE-2023-29007 • CVE-2023-28328 • CVE-2023-27538 • CVE-2023-27536 • CVE-2023-27535 • CVE-2023-27534 • CVE-2023-27533 • CVE-2023-26606 • CVE-2023-26545 • CVE-2023-26544 • CVE-2023-2650 • CVE-2023-2612 • CVE-2023-2603 • CVE-2023-2602 • CVE-2023-25815 • CVE-2023-25652 • CVE-2023-25588 • CVE-2023-25585 • CVE-2023-25584 • CVE-2023-25012 • CVE-2023-23559 • CVE-2023-23455 • CVE-2023-23454 • CVE-2023-2283 • CVE-2023-2162 • CVE-2023-21102 • CVE-2023-20938 • CVE-2023-1998

- CVE-2023-1972
- CVE-2023-1872
- CVE-2023-1859
- CVE-2023-1829
- CVE-2023-1670
- CVE-2023-1667
- CVE-2023-1652
- CVE-2023-1513
- CVE-2023-1380
- CVE-2023-1281
- CVE-2023-1255
- CVE-2023-1079
- CVE-2023-1078
- CVE-2023-1077
- CVE-2023-1076
- CVE-2023-1075
- CVE-2023-1074
- CVE-2023-1073
- CVE-2023-0465
- CVE-2023-0464
- CVE-2023-0459
- CVE-2023-0458
- CVE-2023-0394
- CVE-2023-0386
- CVE-2023-0266
- CVE-2023-0210
- CVE-2023-0045
- CVE-2022-48424
- CVE-2022-48423
- CVE-2022-4842
- CVE-2022-47929
- CVE-2022-4382
- CVE-2022-4129
- CVE-2022-41218
- CVE-2022-3996
- CVE-2022-3707
- CVE-2022-36280
- CVE-2022-3424
- CVE-2022-27672
- CVE-2022-2196

spring-cloud-gateway.tanzu.vmware.com	▼ Expand to see the list • GHSA-crqg-jrpj-fc84 • GHSA-6mjq-h674-j845 • CVE-2023-34035 • CVE-2023-34034 • CVE-2023-33008 • CVE-2023-31484 • CVE-2023-2650 • CVE-2023-2603 • CVE-2023-2602 • CVE-2023-1255

v1.5.4 Known issues

This release has the following known issues, listed by component and area.

v1.5.4 Known issues: Tanzu Application Platform

- Upgrading from Tanzu Application Platform v1.4 to v1.5 sometimes causes temporary failures that self heal in a few minutes. This is because Tanzu Application Platform switched to versioned secrets for all components in v1.5, which can cause a race condition during upgrades and errors similar to the following:

  ```
  Reconcile failed: Preparing template values: secrets "tekton-pipelines-values"
  not found
  ```

v1.5.3

Release Date: 11 July 2023

v1.5.3 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
apis.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-1255 • CVE-2022-3996
buildservice.tanzu.vmware.com	▼ Expand to see the list • GHSA-hw7c-3rfg-p46j
learningcenter.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-2004

Package Name	Vulnerabilities Resolved
sso.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-m8cg-xc2p-r3fc • GHSA-g2j6-57v7-gm8c • GHSA-f3fp-gc8g-vw66 • CVE-2023-2650 • CVE-2023-1255 • CVE-2023-0466 • CVE-2023-0465 • CVE-2023-0464 • CVE-2022-3996 • CVE-2022-3821
workshops.learningcenter.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-2004

v1.5.3 Known issues

This release introduces no new known issues.

v1.5.2

Release Date: 13 June 2023

v1.5.2 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
buildservice.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-1829 • CVE-2023-1281 • CVE-2023-0386
cert-manager.tanzu.vmware.com	▼ Expand to see the list • GHSA-vvpx-j8f3-3w6h
sso.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-31484
tap-gui.tanzu.vmware.com	▼ Expand to see the list • GHSA-f9xv-q969-pqx4

v1.5.2 Resolved issues

The following issues, listed by component and area, are resolved in this release.

v1.5.2 Resolved issues: Supply Chain Security Tools (SCST) - Scan

- Old `TaskRuns` associated with scans are now deleted to reduce memory consumption.

- Added support for `ConfigMaps` in custom `ScanTemplates`.

v1.5.2 Resolved issues: Tanzu Application Platform GUI

- Simplified the default content security policy to remove violations from `fonts.googleapis.com`.

v1.5.2 Resolved issues: Tanzu Application Platform GUI plug-ins

- **Security Analysis GUI plug-in:**
  - **CVE Details:** The impacted workload count in the widget now matches the table.
  - **Security Analysis Dashboard:** The Highest Reach Critical Vulnerabilities chart no longer overlaps Snyk CVE IDs.
  - **Package Details:** Removed extra versions from Workload Builds using Package table.

v1.5.2 Resolved issues: Tanzu Developer Tools for IntelliJ

- Resolved permission-denied errors encountered during Live Update when operating against platforms configured to use the Jammy build stack.

v1.5.2 Resolved issues: Tanzu Developer Tools for Visual Studio

- Resolved permission-denied errors encountered during Live Update when operating against platforms configured to use the Jammy build stack.

v1.5.2 Resolved issues: Tanzu Developer Tools for VS Code

- Resolved permission-denied errors encountered during Live Update when operating against platforms configured to use the Jammy build stack.

---

v1.5.2 Known issues

This release introduces no new known issues.

---

v1.5.1

Release Date: 09 May 2023

v1.5.1 Security fixes

This release has the following security fixes, listed by component and area.

Package Name	Vulnerabilities Resolved
accelerator.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-20860
api-portal.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-20860 • GHSA-493p-pfq6-5258
application-configuration-service.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-20860

Package Name	Vulnerabilities Resolved
apiserver.appliveview.tanzu.vmware.com	▼ Expand to see the list • GHSA-vvpx-j8f3-3w6h • GHSA-r48q-9g5r-8q2h
app-scanning.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-gwc9-m7rh-j2ww • GHSA-fxg5-wq6x-vr4w • GHSA-8c26-wmh5-6g9v • GHSA-69cg-p879-7622 • GHSA-3vm4-22fp-5rfm
backend.appliveview.tanzu.vmware.com	▼ Expand to see the list • GHSA-mjmj-j48q-9wg2 • GHSA-36p3-wjmg-h94x • CVE-2023-20860 • CVE-2022-41881
buildservice.tanzu.vmware.com	▼ Expand to see the list • GHSA-vvpx-j8f3-3w6h • CVE-2023-20860 • CVE-2023-0461
carbonblack.scanning.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-vvpx-j8f3-3w6h • GHSA-vpvm-3wq2-2wvm • GHSA-gwc9-m7rh-j2ww • GHSA-fxg5-wq6x-vr4w • GHSA-8c26-wmh5-6g9v • GHSA-69ch-w2m2-3vjp • GHSA-69cg-p879-7622 • GHSA-3vm4-22fp-5rfm
connector.appliveview.tanzu.vmware.com	▼ Expand to see the list • GHSA-mjmj-j48q-9wg2 • GHSA-36p3-wjmg-h94x • CVE-2023-20860 • CVE-2022-41881
learningcenter.tanzu.vmware.com	▼ Expand to see the list • GHSA-hc6q-2mpp-qw7j • GHSA-frjg-g767-7363 • CVE-2023-26114
metadata-store.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-vvpx-j8f3-3w6h
ootb-templates.tanzu.vmware.com	▼ Expand to see the list • GHSA-vvpx-j8f3-3w6h

Package Name	Vulnerabilities Resolved
scanning.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-vvpx-j8f3-3w6h
snyk.scanning.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-rc47-6667-2j5j • CVE-2023-23919 • CVE-2023-23918
spring-cloud-gateway.tanzu.vmware.com	▼ Expand to see the list • GHSA-493p-pfq6-5258 • CVE-2023-20860
sso.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-0466 • CVE-2023-0465 • CVE-2022-4899
tap-gui.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-0466 • CVE-2023-0465 • CVE-2022-4899
workshops.learningcenter.tanzu.vmware.com	▼ Expand to see the list • GHSA-frjg-g767-7363 • CVE-2023-26114

v1.5.1 Resolved issues

The following issues, listed by component and area, are resolved in this release.

v1.5.1 Resolved issues: Application Accelerator

- The IntelliJ plug-in can now be installed in IntelliJ v2023.1.

v1.5.1 Resolved issues: External Secrets CLI (beta)

- The external-secrets plug-in creating the `ExternalSecret` and `SecretStore` resources through stdin now correctly confirms resource creation. Use `-f` to create resources using a file instead of stdin.

v1.5.1 Resolved issues: Tanzu Developer Tools for IntelliJ

- Live Update now works when using the Jammy `ClusterBuilder`.

v1.5.1 Resolved issues: Tanzu Developer Tools for Visual Studio

- Live Update now works when using the Jammy `ClusterBuilder`.

v1.5.1 Known issues

This release has the following known issues, listed by component and area.

v1.5.1 Known issues: Supply Chain Security Tools (SCST) - Scan

- `TaskRuns` associated with scans are kept during the lifetime of the owner scan. This can lead to Out of Memory restart problems in the SCST - Scan controller.
- `ConfigMaps` used in `ScanTemplates` are not supported, whether introduced by overlays or in a custom `ScanTemplate`. This is the error message you see:

  ```
  The scan job could not be created. admission webhook "validation.webhook.pipeli
  ne.tekton.dev" denied the request: validation failed: expected exactly one, got
  neither: spec.workspaces[5].configmap, spec.workspaces[5].emptydir, spec.worksp
  aces[5].persistentvolumeclaim, spec.workspaces[5].secret, spec.workspaces[5].vo
  lumeclaimtemplate
  ```

v1.5.1 Known issues: Tanzu Application Platform GUI

- Ad-blocking browser extensions and standalone ad-blocking software can interfere with telemetry collection within the VMware Customer Experience Improvement Program and restrict access to all or parts of Tanzu Application Platform GUI. For more information, see Troubleshooting.

---

v1.5.0

Release Date: 11 April 2023

What's new in Tanzu Application Platform

This release includes the following platform-wide enhancements.

v1.5.0 New components

- Application Configuration Service is a new component that provides a Kubernetes-native experience to enable the runtime configuration of existing Spring applications that were previously leveraged by using Spring Cloud Config Server.
- Crossplane is a new component that powers a number of capabilities, such as dynamic provisioning of service instances with Services Toolkit and the Bitnami Services. It is part of the iterate, run, and full profiles.
- Bitnami Services is a new component that includes a set of backing services for Tanzu Application Platform. The provided services are MySQL, PostgreSQL, RabbitMQ and Redis, all of which are backed by the corresponding Bitnami Helm Chart. It is part of the iterate, run, and full profiles.
- Spring Cloud Gateway is an API gateway solution based on the open-source Spring Cloud Gateway project. This new component provides a simple means to route internal or external API requests to application services that expose APIs.

v1.5.0 New features by component and area

This release includes the following changes, listed by component and area.

v1.5.0 Features: Application Accelerator

- The Application Accelerator plug-in for IntelliJ is now available as a beta release on Tanzu Network.

- Adds the option to support Spring Boot v3.0 for the Tanzu Java Restful Web App and Tanzu Java Web App accelerators.
- Application Accelerator now generates application bootstrapping provenance when a project is created using an accelerator. For more information, see Provenance transform.
- Adds the option to use a system-wide property in the `tap-values.yaml` configuration file to activate or deactivate Git repository creation. For more information, see Deactivate Git repository creation.
- The Accelerator Tanzu CLI plug-in now supports using the Tanzu Application Platform GUI URL with the `--server-url` command option. For more information, see Using Tanzu Application Platform GUI URL

v1.5.0 Features: Application Live View

- Application Live View now supports improved security and access control. Introduces the `APIServer` component that generates and validates user access to view actuator data for a pod. For more information, see Improved security and access control in Application Live View.
- Application Live View now supports secure access to sensitive operations that can be executed on a running application using the actuator endpoints at the cluster level. For more information, see Improved security and access control in Application Live View
- The Application Live View plugin now supports CPU stats in the memory and threads pages for Steeltoe Applications. For more information, see Application Live View for Steeltoe Applications in Tanzu Application Platform GUI.

v1.5.0 Features: Application Single Sign-On (AppSSO)

- Introduces `AuthServer` CORS API that enables configuration of allowed HTTP origins. This is useful for public clients, such as single-page apps.
- Introduces an API for filtering external roles, groups, and memberships across OpenID, LDAP, and SAML identity providers in `AuthServer` resource into the `roles` claim of the resulting identity token. For more information, see Roles claim filtering.
- Introduces mapping of user roles, filtered and propagated in the identity token's `roles` claim, into scopes of the access token. For access tokens that are in the JWT format, the resulting scopes are part of the access token's `scope` claim, if the `ClientRegistration` contains the scopes. For more information, see Configure authorization.
- Introduces default access token scopes for user's authentication by using an identity provider. For more information, see Default authorization scopes.
- Introduces standardized client authentication methods to `ClientRegistration` custom resource. For more information, see ClientRegistration.

v1.5.0 Features: Bitnami Services

- The new component Bitnami Services is available with Tanzu Application Platform.
- Provides integration for dynamic provisioning of Bitnami Helm Charts included with Tanzu Application Platform for the following backing services:
    - PostgreSQL
    - MySQL
    - Redis

    - RabbitMQ

  For a tutorial to get started with using these services, see Working with Bitnami Services.

v1.5.0 Features: cert-manager

- `cert-manager.tanzu.vmware.com` has upgraded to cert-manager v1.11.0. For more information, see cert-manager GitHub repository.

v1.5.0 Features: Crossplane

- The new component Crossplane is available with Tanzu Application Platform. It installs Upbound Universal Crossplane v1.11.0.
- Provides integration for dynamic provisioning in Services Toolkit and can be used for integration with cloud services such as AWS, Azure, and GCP. For more information, see Integrating cloud services into Tanzu Application Platform.

  For more information about dynamic provisioning, see Set up dynamic provisioning of service instances to learn more.
- Includes two Crossplane Providers: `provider-kubernetes` and `provider-helm`. You can add other providers manually as required.

v1.5.0 Features: External Secrets CLI (Beta)

- The external-secrets plug-in available in the Tanzu CLI interacts with the External Secrets Operator API. Users can use this CLI plug-in to create and view External Secrets Operator resources on a Kubernetes cluster.

  For more information about managing secrets with External Secrets in general, see the official External Secrets Operator documentation. For installing the External Secrets Operator and the CLI plug-in, see the following documentation:

    - Installing External Secrets Operator in TAP
    - Installing External Secrets CLI plug-in
    - External-Secrets with Hashicorp Vault

  Additionally, see the example integration of External-Secrets with Hashicorp Vault

v1.5.0 Features: Namespace Provisioner

- Includes a new GitOps workflow for managing a list of namespaces fully declaratively through a Git repository. Specify the location of the GitOps repository that has the list of namespaces that you want as ytt data values to be imported in the namespace provisioner using the `gitops_install tap-values.yaml` configuration.

  For more information, see the GitOps section in Provision developer namespaces.
- The Namespace Provisioner controller supports adding namespace parameters from labels or annotations on namespace objects based on accepted prefixes defined in the `parameter_prefixes` configuration in the `tap-values.yaml`. You can use this feature to add custom parameters to a namespace for creating resources conditionally.

  For an example, see Create Tekton pipelines and Scan policies using namespace parameters.
- Adds support for importing Kubernetes secrets that contain a ytt overlay definition that you can apply to the resources that Namespace Provisioner creates.

Using the `overlays_secret` configuration in namespace provisioner `tap-values.yaml`, you can provide a list of secrets that contain the overlay definition to apply to resources created by provisioner.

For an example of using overlays, see Customize OOTB default resources.

- Adds support for reading sensitive data from a Kubernetes secret in YAML format and populating that information in the resources that Namespace Provisioner creates during runtime. This is kept in sync with the source. This removes the need to store any sensitive data in GitOps repository.
    - Using the `import_data_values_secrets` configuration in the Namespace Provisioner section of the Tanzu Application Platform values file, you can import sensitive data from a YAML formatted secret and make it available under `data.values.imported` for additional resource templating.
    - For an example use case, see Install multiple scanners in the developer namespace.
- Namespace Provisioner now creates a Kubernetes `LimitRange` object with acceptable default values that set maximum limits on many resources that pods in the managed namespace can request.
    - Run profile: Stamped by default.
    - Full and iterate profile: Opt-in using parameters.

  For a sample configuration, see Customize OOTB Limit Range default.
- Namespaces Provisioner enables you to use private Git repositories for storing their GitOps based installation files and additional platform operator templated resources that you want to create in your developer namespace. Authentication is provided using a secret in `tap-namespace-provisioning` namespace, or an existing secret in another namespace referred to in the `secretRef` in the additional sources.

  For an example use case, see Working with private Git Repositories

v1.5.0 Features: Services Toolkit

- Services Toolkit now supports the dynamic provisioning of services instances.
    - `ClusterInstanceClass` now supports the new provisioner mode. When a `ClassClaim` is created which refers to a provisioner `ClusterInstanceClass`, a new service instance is created on-demand and claimed. This is powered by Crossplane.
- The `tanzu service` CLI plug-in has the following updates:
    - The command `tanzu service class-claim create` now allows you to pass parameters to the provisioner-based `ClusterInstanceClass` to support dynamic provisioning. For example, `tanzu service class-claim create rmq-claim-1 --class rmq --parameter replicas=3 --parameter ha=true`
    - The `tanzu service class-claim get` now outputs parameters passed as part of claim creation.

  For more information about these commands, see Tanzu Service CLI Plug-In.
- Integrates with the new component Bitnami Services, which provides dynamic provisioning support for the following Helm charts:
    - PostgreSQL
    - MySQL
    - Redis

    - RabbitMQ
- Improves the security model to control which users can claim specific service instances.
    - Introduced the `claim` custom RBAC verb that targets a specific `ClusterInstanceClass`. You can bind this to users for access control of who can create `ClassClaim` resources for a specific `ClusterInstanceClass`.
    - A `ResourceClaimPolicy` is now created automatically for successful `ClassClaims`.

  For more information, see Authorize users and groups to claim from provisioner-based classes to learn more.
- The `ResourceClaimPolicy` now supports targeting individual resources by name. To do so, configure `.spec.subject.resourceNames`.
- The `Where-For-Dinner` sample Application Accelerator now supports dynamic provisioning.
- Changes to the Services Toolkit component documentation.
    - The standalone Services Toolkit documentation is no longer receiving updates. From now on you can find all Services Toolkit documentation in the Tanzu Application Platform component documentation section for Services Toolkit.
    - To learn more about working with services on Tanzu Application Platform, see the new tutorials, how-to guides, concepts, and reference material.

v1.5.0 Features: Supply Chain Choreographer

- Introduces a variation of the Out of the Box Basic supply chains that output Carvel packages. Carvel packages enable configuring for each runtime environment. For more information, see Carvel Package Workflow. This feature is experimental.

v1.5.0 Features: Supply Chain Security Tools (SCST) - Policy Controller

- ClusterImagePolicy resync is triggered every 10 hours to get updated values from the Key Management Service (KMS).

v1.5.0 Features: Supply Chain Security Tools (SCST) - Scan

- SCST - Scan now runs on Tanzu Service Mesh-enabled clusters, enabling end to end, secure communication.
    - Kubernetes jobs that previously created the scan pods were replaced with Tekton TaskRuns.
    - Observability and Troubleshooting documentation is updated to account for the impact of these changes. For successful scans, scanner pods restart once. For more information, see Scanner pod restarts once in SCST - Scan v1.5.0 or later.
- Adds support for rotating certificates and TLS, to conform with NIST 800-53. Users can specify a TLS certificate, minimum TLS version, and restrict TLS ciphers when using kube-rbac-proxy. For more information, see Configure properties.
- SCST - Scan now offers even more flexibility for users to use their existing investments in scanning solutions. In Tanzu Application Platform v1.5.0, users have early access to:
    - A new alpha integration with the Trivy Open Source Vulnerability Scanner by Aqua Security that scans source code and images from secure supply chains. See Install Trivy (alpha).

    - A simplified alpha user experience for creating custom integrations with additional vulnerability scanners that are not included by default. If you have a scanner that you would like to use with Tanzu Application Platform, see SCST - Scan 2.0 (alpha).
    - VMware is looking for early adopters to test both of these alpha offerings and provide feedback. Email your Tanzu representative or contact us here.
- Carbon Black integration is updated to use the Carbon Black scanner CLI v1.9.2. Notable optimizations include improved scan logic that reduces the time it takes for a scan to complete.

  For more information, see the Carbon Black Cloud Console Release Notes.

v1.5.0 Features: Tanzu Application Platform GUI

- **Disclosure:** This upgrade includes a Java script operated by the service provider Pendo.io. The Java script is installed on selected pages of VMware software and collects information about your use of the software, such as clickstream data and page loads, hashed user ID, and limited browser and device information. VMware uses this information to better understand the way you use the software to improve your experience with VMware products and services. For more information, see the Customer Experience Improvement Program.
- Supports automatic configuration with SCST - Store. For more information, see Automatically connect Tanzu Application Platform GUI to the Metadata Store.
- Enables specification of security banners. To use this customization, see Customize security banners.
- Upgrades Backstage to v1.10.1.
- Includes an optional plug-in that collects telemetry by using the Pendo tool. To configure Pendo telemetry and opt in or opt out, see Opt out of telemetry collection.

v1.5.0 Features: Tanzu Application Platform GUI plug-ins

- **Application Live View plug-in:**
    - When `alvToken` has expired, the logic to fetch a new token and the API call are both retried.
    - Actions are deactivated and a message is displayed when sensitive operations are deactivated for the app.
    - The **Heap Dump** button deactivates when sensitive operations are deactivated for the application.
    - Enabled Secure Access Communication between App Live View components.
    - Added an API to connect to `appliveview-apiserver` by reusing `tap-gui` authentication.
    - The Application Live View plug-in now requests a token from `appliveview-apiserver` and passes it to every call to the Application Live View back end.
    - Provides secured sensitive operations (edit env, change log levels, download heap dump) and displays a message in the UI.
    - Renamed the `k8s-logging-backend` plug-in as `k8s-custom-apis-backend`.
    - The fetch token for the `logLevelsPanelToggle` component is now loaded from the workload plug-in PodLogs page.

- **Security Analysis GUI plug-in:**
  - **CVE Details:** Added Impacted Workloads widget to the CVE Details page.
  - **CVE Details:** Display and navigate to latest source SHA or image digest in the Workload Builds table.
  - **Package Details:** Added Impacted Workloads column to the Vulnerabilities table.
  - **Package Details:** Display and navigate to latest source SHA or image digest in the Workload Builds table.
  - **Security Analysis Dashboard:** Added Highest Reach Vulnerabilities widget.

v1.5.0 Features: Tanzu CLI Apps plug-in

- Adds support for `-ojson` and `-oyaml` output flags in `tanzu apps workload create/apply` command. The CLI does not wait to print workload when using `--output` in workload create/apply unless `--wait` or `--tail` flags are specified as well.
- Using the `--no-color` flag in `tanzu apps workload create/apply` commands now hides progress bars in addition to color output and emojis.
- Adds support for unsetting `--git-repo`, `--git-commit`, `--git-tag` and `--git-branch` flags by setting the value to empty string.

v1.5.0 Features: Tanzu Developer Tools for IntelliJ

- Updates the Tanzu Workloads panel to show workloads deployed across multiple namespaces.
- Tanzu actions for `workload apply`, `workload delete`, `debug`, and `Live Update start` are now available from the Tanzu Workloads panel.
- You can use Tanzu Developer Tools for IntelliJ to iterate on Spring Boot 3-based applications.

v1.5.0 Features: Tanzu Developer Tools for Visual Studio

- Supports iterative development of applications consisting of multiple microservices, enabling developers to debug and Live Update each microservice independently and simultaneously.
- Enables existing projects to work with Tanzu Application Platform developer tools easily by using templates to generate the necessary configuration files.

v1.5.0 Features: Tanzu Developer Tools for VS Code

- The Tanzu Activity tab in the Panels view enables developers to visualize the supply chain, delivery, and running application pods.

  The tab enables a developer to view and describe logs on each resource associated with a workload from within their IDE. The tab displays detailed error messages for each resource in an error state.
- Updates the Tanzu Workloads panel to show workloads deployed across multiple namespaces.
- Tanzu commands for `workload apply`, `workload delete`, `debug`, and `Live Update start` are now available from the Tanzu Workloads panel.
- You can use Tanzu Developer Tools for VS Code to iterate on Spring Boot 3-based applications.

v1.5.0 Breaking changes

This release has the following breaking changes, listed by area and component.

v1.5.0 Breaking changes: Convention Controller

- Convention Controller is removed in this release and is replaced by Cartographer Conventions. Cartographer Conventions implements the `conventions.carto.run` API that includes all the features that were available in the Convention Controller component.

v1.5.0 Breaking changes: Supply Chain Security Tools (SCST) - Scan

- The deprecated Grype ScanTemplates included with Tanzu Application Platform v1.2.0 and earlier are removed and no longer supported. Use Grype ScanTemplates v1.2 and later.

v1.5.0 Breaking changes: Tanzu Build Service

- The default `ClusterBuilder` now uses the Ubuntu Jammy v22.04 stack instead of the Ubuntu Bionic v18.04 stack. Previously, the default `ClusterBuilder` pointed to the Base builder based on the Bionic stack. Now, the default `ClusterBuilder` points to the Base builder based on the Jammy stack. Ensure that your workloads can be built and run on Jammy.

  For information about how to change the `ClusterBuilder` from the default builder, see the Configure the Cluster Builder.

  For more information about available builders, see Lite Dependencies and Full Dependencies.
- The Tanzu Build Service automatic dependency updater feature is removed in Tanzu Application Platform v1.5.0. This feature has been deprecated since Tanzu Application Platform v1.2.

v1.5.0 Security fixes

This release has the following security fixes, listed by area and component.

Package Name	Vulnerabilities Resolved
buildservice.tanzu.vmware.com	▼ Expand to see the list • GHSA-fxg5-wq6x-vr4w • CVE-2023-0179
carbonblack.scanning.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-24827
eventing.tanzu.vmware.com	▼ Expand to see the list • GHSA-fxg5-wq6x-vr4w • GHSA-69ch-w2m2-3vjp • GHSA-69cg-p879-7622
grype.scanning.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-24329

Package Name	Vulnerabilities Resolved
learningcenter.tanzu.vmware.com	▼ Expand to see the list • GHSA-x4qr-2fvf-3mr5 • GHSA-ppp9-7jff-5vj2 • GHSA-fxg5-wq6x-vr4w • GHSA-83g2-8m93-v3w7 • GHSA-69ch-w2m2-3vjp • GHSA-3vm4-22fp-5rfm • GHSA-2hrw-hx67-34x6 • CVE-2023-24329 • CVE-2023-23919 • CVE-2023-0461 • CVE-2023-0286
policy.apps.tanzu.vmware.com	▼ Expand to see the list • GHSA-fxg5-wq6x-vr4w
snyk.scanning.apps.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-24329
tap-gui.tanzu.vmware.com	▼ Expand to see the list • CVE-2023-23919 • CVE-2023-23918 • CVE-2023-0361 • CVE-2023-0286 • CVE-2023-0215 • CVE-2022-4450
workshops.learningcenter.tanzu.vmware.com	▼ Expand to see the list • GHSA-ppp9-7jff-5vj2 • GHSA-fxg5-wq6x-vr4w • GHSA-83g2-8m93-v3w7 • GHSA-69ch-w2m2-3vjp • GHSA-3vm4-22fp-5rfm • CVE-2023-24329 • CVE-2023-23919 • CVE-2023-0461 • CVE-2023-0286

Note

CVE-2023-0179, CVE-2023-1281 and CVE-2023-0461 are high severity vulnerabilities. At this time, there is no available patch for them upstream for some Tanzu Application Platform components. After there is a patch available, Tanzu Application Platform will release a patched base stack image. These vulnerabilities are kernel exploits that run on your container host VM, not the Tanzu Application

Platform container image. Running on an up to date kernel is a mitigation for these vulnerabilities.

---

v1.5.0 Resolved issues

The following issues, listed by area and component, are resolved in this release.

v1.5.0 Resolved issues: Application Accelerator

- Resolved issue with `custom types` not re-ordering fields correctly in the VS Code extension.

v1.5.0 Resolved issues: Application Single Sign-On (AppSSO)

- Resolved redirect URI issue with insecure HTTP redirection on Tanzu Kubernetes Grid multicloud (TKGm) clusters.

v1.5.0 Resolved issues: Cloud Native Runtimes

- Resolved issue with DomainMapping names longer than 63 characters when auto-tls is enabled, which is on by default.
- Resolved issue with certain app name, namespace, and domain combinations producing invalid HTTPProxy resources.

v1.5.0 Resolved issues: Namespace Provisioner

- Updated default resources to avoid ownership conflicts with the `grype` package.

v1.5.0 Resolved issues: Tanzu Application Platform GUI plug-ins

- **Application Accelerator plug-in:**
  - Fixed JSON schema for Git repository creation.
  - Added missing query string parameters to accelerator provenance.
- **Application Live View plug-in:**
  - Fixed CPU stats in App Live View Steeltoe Threads and Memory pages.
  - The App Live View Details page now shows the correct boot version instead of **UNKNOWN**.
  - Fixed request parameters for the post-API call.
  - Fixed the UI error in the ALV request-mapping page that was caused by an unused style.
  - Fixed the ALV Request Mappings and Threads page to support Boot 3 apps.

v1.5.0 Resolved issues: Tanzu Build Service

- Builds no longer fail for upgrades on OpenShift v4.11.

v1.5.0 Resolved issues: Tanzu CLI Apps plug-in

- Allow users to pass only `--git-commit` as Git the ref while creating a workload from a Git Repository. This update removes the limitation where users had to provide a `--git-tag` or `--git-branch` with the commit to create a workload.

- Fixed the behavior where `subpath` was getting removed from the workload when there are updates to the Git section of the workload source specification.

v1.5.0 Resolved issues: Tanzu Developer Tools for IntelliJ

- When there are multiple resource types with the same kind, the pop-up menu **Describe** action in the Activity panel no longer fails when used on PodIntent resources.

---

v1.5.0 Known issues

This release has the following known issues, listed by area and component.

v1.5.0 Known issues: API Auto Registration

- Users cannot update their APIs through API Auto Registration due to an issue with the ID used to retrieve APIs. This issue causes errors in the API Descriptor CRD similar to the following: `Unable to find API entity's uid within TAP GUI. Retrying the sync`.

v1.5.0 Known issues: Application Configuration Service

- Client applications that include the `spring-cloud-config-client` dependency might fail to start or properly load the configuration that Application Configuration Service produced.
- Installation might fail because the pod security context does not perfectly adhere to the restricted pod security standard.

v1.5.0 Known issues: Bitnami Services

- If you try to configure private registry integration for the Bitnami services after having already created a claim for one or more of the Bitnami services using the default configuration, the updated private registry configuration does not appear to take effect. This is due to caching behavior in the system which is not accounted for during configuration updates. For a workaround, see Troubleshoot Bitnami Services.

v1.5.0 Known issues: Crossplane

- Crossplane Providers do not transition to `HEALTHY=True` if using a custom certificate for your registry. For more information and a workaround, see Troubleshoot Crossplane.
- Crossplane Providers cannot communicate with systems using a custom CA. For more information and a workaround, see Troubleshoot Crossplane.

v1.5.0 Known issues: Eventing

- When using vSphere sources in Eventing, the vsphere-source is using a high number of informers to alleviate load on the API server. This causes high memory utilization.

v1.5.0 Known issues: External Secrets CLI (beta)

- The external-secrets plug-in creating the `ExternalSecret` and `SecretStore` resources through stdin incorrectly confirms resource creation. Use `-f` to create resources using a file instead of stdin.

v1.5.0 Known issues: Grype scanner

- **Scanning Java source code that uses Gradle package manager might not reveal vulnerabilities:**

  For most languages, source code scanning only scans files present in the source code repository. Except for support added for Java projects using Maven, no network calls fetch dependencies. For languages using dependency lock files, such as golang and Node.js, Grype uses the lock files to verify dependencies for vulnerabilities.

  For Java using Gradle, dependency lock files are not guaranteed, so Grype uses dependencies present in the built binaries, such as `.jar` or `.war` files.

  Grype fails to find vulnerabilities during a source scan because VMware discourages committing binaries to source code repositories. The vulnerabilities are still found during the image scan after the binaries are built and packaged as images.

v1.5.0 Known issues: Services Toolkit

- Unexpected error if `additionalProperties` is `true` in a CompositeResourceDefinition. For more information and a workaround, see Troubleshoot Services Toolkit.
- Default cluster-admin IAM roles on GKE do not allow you to claim Bitnami Services. For more information and a workaround, see Troubleshoot Services Toolkit.

v1.5.0 Known issues: Supply Chain Choreographer

- When using the Carvel Package Supply Chains, if the operator updates the parameter `carvel_package.name_suffix`, existing workloads incorrectly output a Carvel package to the GitOps repository that uses the old value of `carvel_package.name_suffix`. You can ignore or delete this package.

v1.5.0 Known issues: Tanzu Application Platform GUI

- The portal might partially overlay text on the Security Banners customization at the bottom.
- The **Impacted Workloads** table is empty on the **CVE and Package Details** pages if the relevant CVE belongs to a workload that has only completed one type of vulnerability scan (either image or source). A fix is planned for a later patch.

v1.5.0 Known issues: Tanzu CLI Apps plug-in

- `tanzu apps workload apply` does not wait for the changes to be taken when the workload is updated using `--tail` or `--wait`. Instead it fails if the status before the changes shows an error.

v1.5.0 Known issues: Tanzu Developer Tools for IntelliJ

- The error `com.vdurmont.semver4j.SemverException: Invalid version (no major version)` is shown in the error logs when attempting to perform a workload action before installing the Tanzu CLI apps plug-in.
- The `apply` action prompts and stores the workload file path when using the action for the first time, but modifying it afterwards is not possible. If the workload file location changes you must delete the module's key-value entries to delete the configuration. These entries are prefixed with `com.tanzu` in `PropertiesComponent` in the project's `.idea/workspace.xml` file. The next `apply` action run prompts for new values again.
- If you restart your computer while running Live Update without terminating the Tilt process beforehand, there is a lock that incorrectly shows that Live Update is still running and

prevents it from starting again. To resolve this, delete the Tilt lock file. The default location for the file is `~/.tilt-dev/config.lock`.

- On Windows, workload actions do not work when in a project with spaces in the name such as `my-app project`. For more information, see Troubleshooting.
- In the Tanzu Activity Panel, the `config-writer-pull-requester` of type `Runnable` is incorrectly categorized as **Unknown**. The correct category is **Supply Chain**.

v1.5.0 Known issues: Tanzu Developer Tools for Visual Studio

- Clicking the red square Stop button in the Visual Studio top toolbar can cause a workload to fail. For more information, see Troubleshooting.

v1.5.0 Known issues: Tanzu Developer Tools for VS Code

- If you restart your computer while running Live Update without terminating the Tilt process beforehand, there is a lock that incorrectly shows that Live Update is still running and prevents it from starting again. Delete the Tilt lock file to resolve this. The default file location is `~/.tilt-dev/config.lock`.
- On Windows, workload commands don't work when in a project with spaces in the name, such as `my-app project`. For more information, see Troubleshooting.
- If your kubeconfig file `~/.kube/config` is malformed, you cannot apply a workload. You see an error message when you attempt to do so. To resolve this, fix the kubeconfig file.
- In the Tanzu Activity Panel, the `config-writer-pull-requester` of type `Runnable` is incorrectly categorized as **Unknown**. The correct category is **Supply Chain**.

v1.5.0 Known issues: Tanzu Source Controller

- In v0.7.0, when pulling images from Elastic Container Registry (ECR), Tanzu Source Controller keyless access to ECR through AWS IAM role binding fails to authenticate (error code: 401). The workaround is to set up a standard Kubernetes secret with a user-id and password to authenticate to ECR, instead of binding Tanzu Source Controller to an AWS IAM role to pull images from ECR.

Deprecations

The following features, listed by component, are deprecated. Deprecated features will remain on this list until they are retired from Tanzu Application Platform.

Application Live View deprecations

- `appliveview_connnector.backend.sslDisabled` is deprecated and marked for removal in Tanzu Application Platform v1.7.0. For more information about the migration, see Deprecate the sslDisabled key.

Application Single Sign-On (AppSSO) deprecations

- `ClientRegistration` resource `clientAuthenticationMethod` field values `post` and `basic` are deprecated and marked for removal in Tanzu Application Platform v1.7.0. Use `client_secret_post` and `client_secret_basic` instead.
- `AuthServer.spec.tls.disabled` is deprecated and marked for removal in Tanzu Application Platform v1.6.0. For more information about how to migrate to `AuthServer.spec.tls.deactivated`, see Migration guides.

Services Toolkit deprecations

- The `tanzu services claims` CLI plug-in command is now deprecated. It is hidden from help text output, but continues to work until officially removed after the deprecation period. The new `tanzu services resource-claims` command provides the same function.

Supply Chain Security Tools (SCST) - Scan deprecations

- The `docker` field and related sub-fields used in SCST - Scan are deprecated and marked for removal in Tanzu Application Platform v1.7.0.

  The deprecation impacts the following components: Scan Controller, Grype Scanner, and Snyk Scanner. Carbon Black Scanner is not impacted. For information about the migration path, see Troubleshooting.

Tanzu Build Service deprecations

- The Ubuntu Bionic stack is deprecated: Ubuntu Bionic stops receiving support in April 2023. VMware recommends you migrate builds to Jammy stacks in advance. For how to migrate builds, see Use Jammy stacks for a workload.
- The Cloud Native Buildpack Bill of Materials (CNB BOM) format is deprecated. VMware plans to deactivate this format by default in Tanzu Application Platform v1.6.1 and remove support in Tanzu Application Platform v1.8. To manually deactivate legacy CNB BOM support, see Deactivate the CNB BOM format.

Tanzu CLI Apps plug-in deprecations

- The default value for the –update-strategy flag will change from `merge` to `replace` in Tanzu Application Platform v1.7.0.
- The `tanzu apps workload update` command is deprecated and marked for removal in Tanzu Application Platform v1.6.0. Use the command `tanzu apps workload apply` instead.

Linux Kernel CVEs

Kernel level vulnerabilities are regularly identified and patched by Canonical. Tanzu Application Platform releases with available images, which might contain known vulnerabilities. When Canonical makes patched images available, Tanzu Application Platform incorporates these fixed images into future releases.

The kernel runs on your container host VM, not the Tanzu Application Platform container image. Even with a patched Tanzu Application Platform image, the vulnerability is not mitigated until you deploy your containers on a host with a patched OS. An unpatched host OS might be exploitable if the base image is deployed.

Components and installation profiles for Tanzu Application Platform

This topic lists the components you can install with Tanzu Application Platform (commonly known as TAP). You can install components as individual packages or you can install them using a profile containing a predefined group of packages.

Tanzu Application Platform components

- **API Auto Registration**

  When users deploy a workload that exposes an API, they want that API to automatically show in Tanzu Application Platform GUI without requiring any added manual steps.

  API Auto Registration is an automated workflow that can use a supply chain to create and manage a Kubernetes Custom Resource (CR) of type `APIDescriptor`. A Kubernetes controller reconciles the CR and updates the API entity in Tanzu Application Platform GUI to achieve automated API registration from workloads. You can also use API Auto Registration without supply chains by directly applying an `APIDescriptor` CR to the cluster.

- **API portal**

  API portal for VMware Tanzu enables API consumers to find APIs they can use in their own applications.

  Consumers can view detailed API documentation and try out an API to see if it meets their needs. API portal assembles its dashboard and detailed API documentation views by ingesting OpenAPI documentation from the source URLs. An API portal operator can add any number of OpenAPI source URLs to appear in a single instance.

- **API Scoring and Validation**

  API Validation and Scoring focuses on scanning and validating an OpenAPI specification. The API specification is generated from the API Auto Registration. After an API is registered, the API specification goes through static scan analysis and is validated. Based on the validation, a scoring is provided to indicate the quality and health of the API specification as it relates to Documentation, OpenAPI best practices, and Security.

- **Application Accelerator**

  The Application Accelerator component helps app developers and app operators create application accelerators.

  Accelerators are templates that codify best practices and ensure that important configurations and structures are in place. Developers can bootstrap their applications and get started with feature development right away.

  Application operators can create custom accelerators that reflect their desired architectures and configurations and enable fleets of developers to use them. This helps ease operator concerns about whether developers are implementing their best practices.

- **Application Configuration Service**

Application Configuration Service provides a Kubernetes-native experience to enable the runtime configuration of existing Spring applications that were previously leveraged by using Spring Cloud Config Server.

Application Configuration Service is compatible with the existing Git repository configuration management approach. It filters runtime configuration for any application by using slices that produce secrets.

- **Application Live View**

  Application Live View is a lightweight insight and troubleshooting tool that helps application developers and application operators look inside running applications.

  It is based on the concept of Spring Boot Actuators. The application provides information from inside the running processes by using endpoints (in our case, HTTP endpoints). Application Live View uses those endpoints to get the data from the application and to interact with it.

- **Application Single Sign-On**

  Application Single Sign-On enables application users to sign in to their identity provider once and be authorized and identified to access any Kubernetes-deployed workload. It is a secure and straightforward approach for developers and operators to manage access across all workloads in the enterprise.

- **Bitnami Services**

  Bitnami Services provides a set of services for Tanzu Application Platform backed by corresponding Bitnami Helm Charts. Through integration with Crossplane and Services Toolkit, these Bitnami Services are immediately ready for apps teams to consume, with no additional setup or configuration required from ops teams. This makes it incredibly quick and easy to get started working with services on Tanzu Application Platform.

- **Cartographer Conventions**

  Use Cartographer Conventions to ensure infrastructure uniformity across workloads deployed on the cluster. Cartographer Conventions provide a way to control how applications should be deployed on Kubernetes using a convention. Use Cartographer Conventions to apply the runtime best practices, policies, and conventions of your organization to workloads as they are created on the platform.

- **cert-manager**

  cert-manager adds certificates and certificate issuers as resource types to Kubernetes clusters. It also helps you to obtain, renew, and use those certificates. For more information about cert-manager, see the cert-manager documentation.

- **Cloud Native Runtimes**

  Cloud Native Runtimes for Tanzu is a serverless application runtime for Kubernetes that is based on Knative and runs on a single Kubernetes cluster. For information about Knative, see the Knative documentation.

- **Contour**

  Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multi-team ingress delegation. It provides the control plane for the Envoy edge and service proxy. For more information about Contour, see the Contour documentation.

- **Default roles for Tanzu Application Platform**

  This package includes five default roles for users, including app-editor, app-viewer, app-operator, and service accounts including workload and deliverable. These roles are available to help operators limit permissions a user or service account requires on a cluster that runs

Tanzu Application Platform. They are built by using aggregated cluster roles in Kubernetes role-based access control (RBAC). Default roles only apply to a user interacting with the cluster by using kubectl and Tanzu CLI.

- **Crossplane**

  Crossplane is an open source, Cloud Native Computing Foundation (CNCF) project built on the foundation of Kubernetes. Tanzu Application Platform uses Crossplane to power a number of capabilities, such as dynamic provisioning of services instances with Services Toolkit and the Bitnami Services.

- **Developer Conventions**

  Developer conventions configure workloads to prepare them for inner loop development.

  It's meant to be a "deploy and forget" component for developers. After it is installed on the cluster with the Tanzu Package CLI, developers do not need to directly interact with it. Developers instead interact with the Tanzu Developer Tools for VSCode IDE Extension or Tanzu CLI Apps plug-in, which rely on the Developer Conventions to edit the workload to enable inner loop capabilities.

- **Eventing**

  Eventing for VMware Tanzu focuses on providing tooling and patterns for Kubernetes applications to manage event-triggered systems through Knative Eventing. For information about Knative, see the Knative documentation.

- **Flux CD Source Controller**

  The main role of this source management component is to provide a common interface for artifact acquisition.

- **Learning Center**

  Learning Center provides a platform for creating and self-hosting workshops. With Learning Center, content creators can create workshops from markdown files that learners can view in a terminal shell environment with an instructional wizard UI. The UI can embed slide content, an integrated development environment (IDE), a web console for accessing the Kubernetes cluster, and other custom web applications.

  Although Learning Center requires Kubernetes to run, and it teaches users about Kubernetes, you can use it to host training for other purposes as well. For example, you can use it to train users on web-based applications, use of databases, or programming languages.

- **Namespace Provisioner**

  Namespace Provisioner provides an easy, secure, automated way for Platform Operators to provision namespaces with the resources and proper namespace-level privileges needed for developer workloads to function as intended.

- **Service Bindings**

  Service Bindings create a Kubernetes-wide specification for communicating service secrets to workloads in a consistent way.

- **Services Toolkit**

  Services Toolkit is responsible for backing many of the most exciting and powerful capabilities for services in Tanzu Application Platform. From the integration of an extensive list of cloud-based and on-prem services, through to the offering and discovery of those services, and finally to the claiming and binding of service instances to application workloads, Services Toolkit has the tools you need to make working with services on Tanzu Application Platform simple, easy, and effective.

- **Source Controller**

  Tanzu Source Controller provides a standard interface for artifact acquisition and extends the function of Flux CD Source Controller. Tanzu Source Controller supports the following two resource types:

  - ImageRepository
  - MavenArtifact

- **Spring Boot conventions**

  The Spring Boot convention server has a bundle of smaller conventions applied to any Spring Boot application that is submitted to the supply chain in which the convention controller is configured.

- **Spring Cloud Gateway**

  Spring Cloud Gateway for Kubernetes is an API gateway solution based on the open-source Spring Cloud Gateway project. It provides a simple means to route internal or external API requests to application services that expose APIs.

- **Supply Chain Choreographer**

  Supply Chain Choreographer is based on open-source Cartographer. It enables app operators to create preapproved paths to production by integrating Kubernetes resources with the elements of their existing toolchains, such as Jenkins.

  Each pre-approved supply chain creates a paved road to production. It orchestrates supply chain resources, namely test, build, scan, and deploy. Enabling developers to focus on delivering value to their users. Pre-approved supply chains also assure application operators that all code in production has passed through the steps of an approved workflow.

- **Supply Chain Security Tools - Policy Controller**

  Supply Chain Security Tools - Policy is an admission controller that allows a cluster operator to specify policies to verify image container signatures before admitting them to a cluster. It works with cosign signature format and allows for fine-tuned configuration of policies based on image source patterns.

- **Supply Chain Security tools for Tanzu - Scan**

  With Supply Chain Security Tools for VMware Tanzu - Scan, you can build and deploy secure trusted software that complies with their corporate security requirements.

  To enable this, Supply Chain Security Tools - Scan provides scanning and gate keeping capabilities that Application and DevSecOps teams can incorporate earlier in their path to production. This is an established industry best practice for reducing security risk and ensuring more efficient remediation.

- **Supply Chain Security Tools - Store**

  Supply Chain Security Tools - Store saves software bills of materials (SBoMs) to a database and enables you to query for image, source, package, and vulnerability relationships. It integrates with SCST - Scan to automatically store the resulting source and image vulnerability reports.

- **Tanzu Application Platform GUI**

  Tanzu Application Platform GUI lets your developers view your organization's running applications and services. It provides a central location for viewing dependencies, relationships, technical documentation, and even service status. Tanzu Application Platform GUI is built from the Cloud Native Computing Foundation's project Backstage.

- **Tanzu Application Platform Telemetry**

Tanzu Application Platform Telemetry is a set of objects that collect data about the use of Tanzu Application Platform and send it back to VMware for product improvements. A benefit of remaining enrolled in telemetry and identifying your company during Tanzu Application Platform installation is that VMware can provide your organization with use reports about Tanzu Application Platform. For information about enrolling in telemetry reports, see Tanzu Application Platform usage reports.

> **Note**
>
> You can opt out of telemetry collection by following the instructions in Opting out of telemetry collection.

- **Tanzu Build Service**

  Tanzu Build Service uses the open-source Cloud Native Build packs project to turn application source code into container images.

  Tanzu Build Service executes reproducible builds that align with modern container standards and keeps images up to date. It does so by leveraging Kubernetes infrastructure with kpack, a Cloud Native Build packs Platform, to orchestrate the image life cycle.

  The kpack CLI tool, kp, can aid in managing kpack resources. Build Service helps you develop and automate containerized software workflows securely and at scale.

- **Tanzu Developer Tools for IntelliJ**

  Tanzu Developer Tools for IntelliJ is the official VMware Tanzu IDE extension for IntelliJ IDEA to help you develop code by using Tanzu Application Platform. This extension enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

- **Tanzu Developer Tools for Visual Studio**

  Tanzu Developer Tools for Visual Studio is the official VMware Tanzu IDE extension for Visual Studio to help you develop code by using Tanzu Application Platform. The Visual Studio extension enables live updates of your application while it runs on the cluster and lets you debug your application directly on the cluster.

- **Tanzu Developer Tools for Visual Studio Code**

  Tanzu Developer Tools for VS Code is the official VMware Tanzu IDE extension for VS Code to help you develop code by using Tanzu Application Platform. The VS Code extension enables live updates of your application while it runs on the cluster and lets you debug your application directly on the cluster.

- **Tekton Pipelines**

  Tekton is a powerful and flexible open-source framework for creating CI/CD systems, enabling developers to build, test, and deploy across cloud providers and on-premise systems.

Installation profiles in Tanzu Application Platform v1.5

You can deploy Tanzu Application Platform through predefined profiles, each containing various packages, or you can install the packages individually. The profiles allow Tanzu Application Platform to scale across an organization's multicluster, multi-cloud, or hybrid cloud infrastructure. These profiles are not meant to cover all use cases, but serve as a starting point to allow for further customization.

The following profiles are available in Tanzu Application Platform:

- **Full** (`full`): Contains nearly all Tanzu Application Platform packages. For the exceptions to the full profile, see the packages with a check mark in the **Not in a profile** column in the table later in this section.
- **Iterate** (`iterate`): Intended for iterative application development.
- **Build** (`build`): Intended for the transformation of source revisions to workload revisions. Specifically, hosting workloads and SupplyChains.
- **Run** (`run`): Intended for the transformation of workload revisions to running pods. Specifically, hosting deliveries and deliverables.
- **View** (`view`): Intended for instances of applications related to centralized developer experiences. Specifically, Tanzu Application Platform GUI and Metadata Store.

The following tables list the packages contained in each profile. Packages not included in any profile are available to install as individual packages only. See the component documentation for the package for installation instructions. For a diagram showing the packages contained in each profile, see Overview of multicluster Tanzu Application Platform.

Packages: A to C

Package Name	Full	Iterate	Build	Run	View	Not in a profile
API Auto Registration	✓	✓		✓		
API portal	✓				✓	
Application Accelerator	✓				✓	
Application Configuration Service						✓
Application Live View APIServer	✓	✓		✓		
Application Live View back end	✓				✓	
Application Live View connector	✓	✓		✓		
Application Live View conventions	✓	✓	✓			
Application Single Sign-On	✓	✓		✓		
Bitnami Services	✓	✓		✓		
Carbon Black Scanner for SCST - Scan (beta)						✓
cert-manager	✓	✓	✓	✓	✓	
Cloud Native Runtimes	✓	✓		✓		
Contour	✓	✓	✓	✓	✓	
Crossplane	✓	✓		✓		

Packages: D to R

Package Name	Full	Iterate	Build	Run	View	Not in a profile
Default Roles	✓	✓	✓	✓		
Developer Conventions	✓	✓				
External Secrets Operator						✓
Eventing	✓	✓		✓		
Flux Source Controller	✓	✓	✓	✓	✓	

Package Name	Full	Iterate	Build	Run	View	Not in a profile
Grype Scanner for SCST - Scan	✓		✓			
Learning Center	✓				✓	
Namespace Provisioner	✓	✓	✓	✓		
Out of the Box Delivery - Basic	✓	✓		✓		
Out of the Box Supply Chain - Basic	✓	✓	✓			
Out of the Box Supply Chain - Testing	✓	✓	✓			
Out of the Box Supply Chain - Testing and Scanning	✓		✓			
Out of the Box Templates	✓	✓	✓	✓		

Packages: S to Z

Package Name	Full	Iterate	Build	Run	View	Not in a profile
Service Bindings	✓	✓		✓		
Services Toolkit	✓	✓		✓		
Source Controller	✓	✓	✓	✓	✓	
Snyk Scanner for SCST - Scan (beta)						✓
Spring Boot conventions	✓	✓	✓			
Spring Cloud Gateway						✓
Supply Chain Choreographer	✓	✓	✓	✓		
SCST - Policy Controller	✓	✓		✓		
SCST - Scan	✓		✓			
SCST - Scan 2.0 (beta)						✓
SCST - Store	✓				✓	
Tanzu Build Service	✓	✓	✓			
Tanzu Application Platform GUI	✓				✓	
Tekton Pipelines	✓	✓	✓			
Telemetry	✓	✓	✓	✓	✓	

Note

You can only install one supply chain at any given time. For information about switching supply chains, see Add testing and scanning to your application.

Language and framework support in Tanzu Application Platform

The following table shows the languages and frameworks supported by Tanzu Application Platform components.

Language or Framework	Tanzu Build Service	Runtime Conventions	Tanzu Developer Tooling	Application Live View	Functions	Extended Scanning Coverage using Buildpack SBOM's
Java	✓	✓	✓		✓	✓
Spring Boot	✓	✓	✓	✓	✓	✓
.NET Core	✓		✓	✓		✓
Steeltoe	✓	✓	✓	✓		
NodeJS	✓				✓	✓
Python	✓				✓	✓
golang	✓					✓
PHP	✓					✓
Ruby	✓					✓

Tanzu Developer Tooling: refers to the developer conventions that enable debugging and Live Update function in the inner loop.

Extended Scanning Coverage: When building container images with the Tanzu Build Service, the Cloud Native Build Packs used in the build process for the specified languages produce a Software Bill of Materials (SBOM). Some scan engines support the enhanced ability to use this SBOM as a source for the scan. Out of the Box Supply Chain - Testing and Scanning leverages Anchore's Grype for the image scan, which suppports this capability. In addition, users have the ability to leverage Carbon Black Container image scans, which also supports this enhanced scan coverage.

Note: Different scanners may have different limits. See Supported Scanner Matrix for Supply Chain Security Tools - Scan.

Installing Tanzu Application Platform

For more information about installing Tanzu Application Platform, see Installing Tanzu Application Platform.

Install Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) by using one of the following methods:

- Install Tanzu Application Platform online. For Tanzu Application Platform on a Kubernetes cluster with internet access.
- Install Tanzu Application Platform in an air-gapped environment. For Tanzu Application Platform on a Kubernetes cluster air-gapped from external traffic.
- Install Tanzu Application Platform with GitOps (beta). For Tanzu Application Platform on a Kubernetes cluster via a GitOps approach.
- Install Tanzu Application Platform in AWS. For installing Tanzu Application platform using AWS Cloud Services.
- Install Tanzu Application Platform in Azure. For installing Tanzu Application platform using Azure Cloud Services.
- Install Tanzu Application Platform on OpenShift. For Tanzu Application Platform on an OpenShift cluster with internet access.

Install Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) by using one of the following methods:

- Install Tanzu Application Platform online. For Tanzu Application Platform on a Kubernetes cluster with internet access.
- Install Tanzu Application Platform in an air-gapped environment. For Tanzu Application Platform on a Kubernetes cluster air-gapped from external traffic.
- Install Tanzu Application Platform with GitOps (beta). For Tanzu Application Platform on a Kubernetes cluster via a GitOps approach.
- Install Tanzu Application Platform in AWS. For installing Tanzu Application platform using AWS Cloud Services.
- Install Tanzu Application Platform in Azure. For installing Tanzu Application platform using Azure Cloud Services.
- Install Tanzu Application Platform on OpenShift. For Tanzu Application Platform on an OpenShift cluster with internet access.

Prerequisites for installing Tanzu Application Platform

The following are required to install Tanzu Application Platform (commonly known as TAP):

VMware Tanzu Network and container image registry requirements

Installation requires:

- Access to VMware Tanzu Network:
  - A Tanzu Network account to download Tanzu Application Platform packages.
  - Network access to https://registry.tanzu.vmware.com.
- Cluster-specific registry:
  - A container image registry, such as Harbor or Docker Hub for application images, base images, and runtime dependencies. When available, VMware recommends using a paid registry account to avoid potential rate-limiting associated with some free registry offerings.
  - Recommended storage space for container image registry:
    - 1 GB of available storage if installing Tanzu Build Service with the `lite` set of dependencies.
    - 10 GB of available storage if installing Tanzu Build Service with the `full` set of dependencies, which are suitable for offline environments.

    Note

    For production environments, `full` dependencies are recommended to optimize security and performance. For more information about Tanzu Build Service dependencies, see About lite and full dependencies.
- Registry credentials with read and write access available to Tanzu Application Platform to store images.
- Network access to your chosen container image registry.

DNS Records

There are some optional but recommended DNS records you must allocate if you decide to use these particular components:

- Cloud Native Runtimes (Knative): Allocate a wildcard subdomain for your developer's applications. This is specified in the `shared.ingress_domain` key of the `tap-values.yaml` configuration file that you input with the installation. This wildcard must be pointed at the external IP address of the `tanzu-system-ingress`'s `envoy` service. See Access with the shared Ingress method for more information about `tanzu-system-ingress`.
- Tanzu Learning Center: Similar to Cloud Native Runtimes, allocate a wildcard subdomain for your workshops and content. This is also specified by the `shared.ingress_domain` key of the `tap-values.yaml` configuration file that you input with the installation. This wildcard must be pointed at the external IP address of the `tanzu-system-ingress`'s `envoy` service.
- Tanzu Application Platform GUI: If you decide to implement the shared ingress and include Tanzu Application Platform GUI, allocate a fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `tap-gui` and the `shared.ingress_domain` value. For example, `tap-gui.example.com`.
- Supply Chain Security Tools - Store: Similar to Tanzu Application Platform GUI, allocate a fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `metadata-store` and the `shared.ingress_domain` value. For example, `metadata-store.example.com`.

- Application Live View: If you select the `ingressEnabled` option, allocate a corresponding fully Qualified Domain Name (FQDN) that can be pointed at the `tanzu-system-ingress` service. The default host name consists of `appliveview` and the `shared.ingress_domain` value. For example, `appliveview.example.com`.

Tanzu Application Platform GUI

For Tanzu Application Platform GUI, you must have:

- Latest version of Chrome, Firefox, or Edge. Tanzu Application Platform GUI currently does not support Safari browser.
- Git repository for Tanzu Application Platform GUI's software catalogs, with a token allowing read access. For more information about how to use your Git repository, see Create an application accelerator. Supported Git infrastructure includes:
    - GitHub
    - GitLab
    - Azure DevOps
- Tanzu Application Platform GUI Blank Catalog from the Tanzu Application section of VMware Tanzu Network.
    - To install, navigate to Tanzu Network. Under the list of available files to download, there is a folder titled `tap-gui-catalogs-latest`. Inside that folder is a compressed archive titled `Tanzu Application Platform GUI Blank Catalog`. You must extract that catalog to the preceding Git repository of choice. This serves as the configuration location for your organization's catalog inside Tanzu Application Platform GUI.
- The Tanzu Application Platform GUI catalog allows for two approaches to store catalog information:
    - The default option uses an in-memory database and is suitable for test and development scenarios. This reads the catalog data from Git URLs that you specify in the `tap-values.yaml` file. This data is temporary. Any operations that cause the `server` pod in the `tap-gui` namespace to be re-created also cause this data to be rebuilt from the Git location. This can cause issues when you manually register entities by using the UI, because they only exist in the database and are lost when that in-memory database gets rebuilt.
    - For production use cases, use a PostgreSQL database that exists outside the Tanzu Application Platform packaging. The PostgreSQL database stores all the catalog data persistently both from the Git locations and the UI manual entity registrations. For more information, see Configure the Tanzu Application Platform GUI database

Kubernetes cluster requirements

Installation requires Kubernetes cluster v1.24, v1.25 or v1.26 on one of the following Kubernetes providers:

- Azure Kubernetes Service.
- Amazon Elastic Kubernetes Service.
    - containerd must be used as the Container Runtime Interface (CRI). Some versions of EKS default to Docker as the container runtime and must be changed to containerd.
    - EKS clusters on Kubernetes version 1.23 and above require the Amazon EBS CSI Driver due to CSIMigrationAWS is enabled by default in Kubernetes version 1.23 and above.

        - Users currently on EKS Kubernetes version 1.22 must install the Amazon EBS CSI Driver before upgrading to Kubernetes version 1.23 and above. See AWS documentation for more information.
    - AWS Fargate is not supported.
- Google Kubernetes Engine.
    - GKE Autopilot clusters do not have the required features enabled.
    - GKE clusters that are set up in zonal mode might detect Kubernetes API errors when the GKE control plane is resized after traffic increases. Users can mitigate this by creating a regional cluster with three control-plane nodes right from the start.
- Minikube.
    - Reference the resource requirements in the following section.
    - Hyperkit driver is supported on macOS only. Docker driver is not supported.
- Red Hat OpenShift Container Platform v4.11 or v4.12.
    - vSphere
    - Baremetal
- Tanzu Kubernetes Grid multicloud.
- vSphere with Tanzu v8.0.1 or later.
    - For vSphere with Tanzu, you must configure pod security policies so Tanzu Application Platform controller pods can run as root. For more information, see Kubernetes documentation.

      To set the pod security policies, run:

      ```
      kubectl create clusterrolebinding default-tkg-admin-privileged-binding --
      clusterrole=psp:vmware-system-privileged --group=system:authenticated
      ```

      For more information about pod security policies on Tanzu for vSphere, see VMware vSphere Product Documentation.

For more information about the supported Kubernetes versions, see Kubernetes version support for Tanzu Application Platform.

Resource requirements

- To deploy Tanzu Application Platform packages iterate profile on local Minikube cluster, your cluster must have at least:
    - 8 vCPUs for i9 (or equivalent) available to Tanzu Application Platform components on Mac OS.
    - 12 vCPUs for i7 (or equivalent) available to Tanzu Application Platform components on Mac OS.
    - 8 vCPUs available to Tanzu Application Platform components on Linux and Windows.
    - 12 GB of RAM available to Tanzu Application Platform components on Mac OS, Linux and Windows.
    - 70 GB of disk space available per node.
- To deploy Tanzu Application Platform packages full profile, your cluster must have at least:
    - 8 GB of RAM available per node to Tanzu Application Platform.
    - 16 vCPUs available across all nodes to Tanzu Application Platform.

    - 100 GB of disk space available per node.
- To deploy Tanzu Application Platform packages build, run and iterate (shared) profile, your cluster must have at least:
    - 8 GB of RAM available per node to Tanzu Application Platform.
    - 12 vCPUs available across all nodes to Tanzu Application Platform.
    - 100 GB of disk space available per node.
- To deploy Tanzu Application Platform packages view profile, your cluster must have at least:
    - 8 GB of RAM available per node to Tanzu Application Platform.
    - 8 vCPUs available across all nodes to Tanzu Application Platform.
    - 100 GB of disk space available per node.
- For the full profile or use of Security Chain Security Tools - Store, your cluster must have a configured default StorageClass.
- Pod security policies must be configured so that Tanzu Application Platform controller pods can run as root in the following optional configurations:
    - Tanzu Build Service, in which CustomStacks require root privileges. For more information, see Tanzu Build Service documentation.
    - Supply Chain, in which Kaniko usage requires root privileges to build containers.
    - Tanzu Learning Center, which requires root privileges.

  For more information about pod security policies, see Kubernetes documentation.

Tools and CLI requirements

Installation requires:

- The Kubernetes CLI (kubectl) v1.24, v1.25, or v1.26 installed and authenticated with admin rights for your target cluster. See Install Tools in the Kubernetes documentation.

Next steps

- Accept Tanzu Application Platform EULAs and installing the Tanzu CLI

Kubernetes version support for Tanzu Application Platform

The following is a matrix table providing details of the compatible Kubernetes cluster versions for Tanzu Application Platform v1.5.

Kubernetes Cluster	Support Information	Notes
Kubernetes	v1.24, v1.25, v1.26	
VMware Tanzu Kubernetes Grid	v2.3.0, v2.2.0, v2.1.1, v2.1.0	Support for Tanzu Kubernetes Grid v2.3.x begins with Tanzu Application Platform v1.5.4 Support for Tanzu Kubernetes Grid v2.2.x begins with Tanzu Application Platform v1.5.2
Tanzu Kubernetes releases (vSphere with Tanzu)	TKr v1.25.7 for vSphere v8.x, TKr v1.24.9 for vSphere v8.x	Support for TKr v1.25.7 begins with Tanzu Application Platform v1.5.4
OpenShift	v4.11, v4.12	

Kubernetes Cluster	Support Information	Notes
Azure Kubernetes Service	Supported	
Elastic Kubernetes Service	Supported	
Google Kubernetes Engine	Supported	

Install Tanzu CLI

This topic tells you how to accept the EULAs, and install the Tanzu CLI and plug-ins on Tanzu Application Platform (commonly known as TAP).

Accept the End User License Agreements

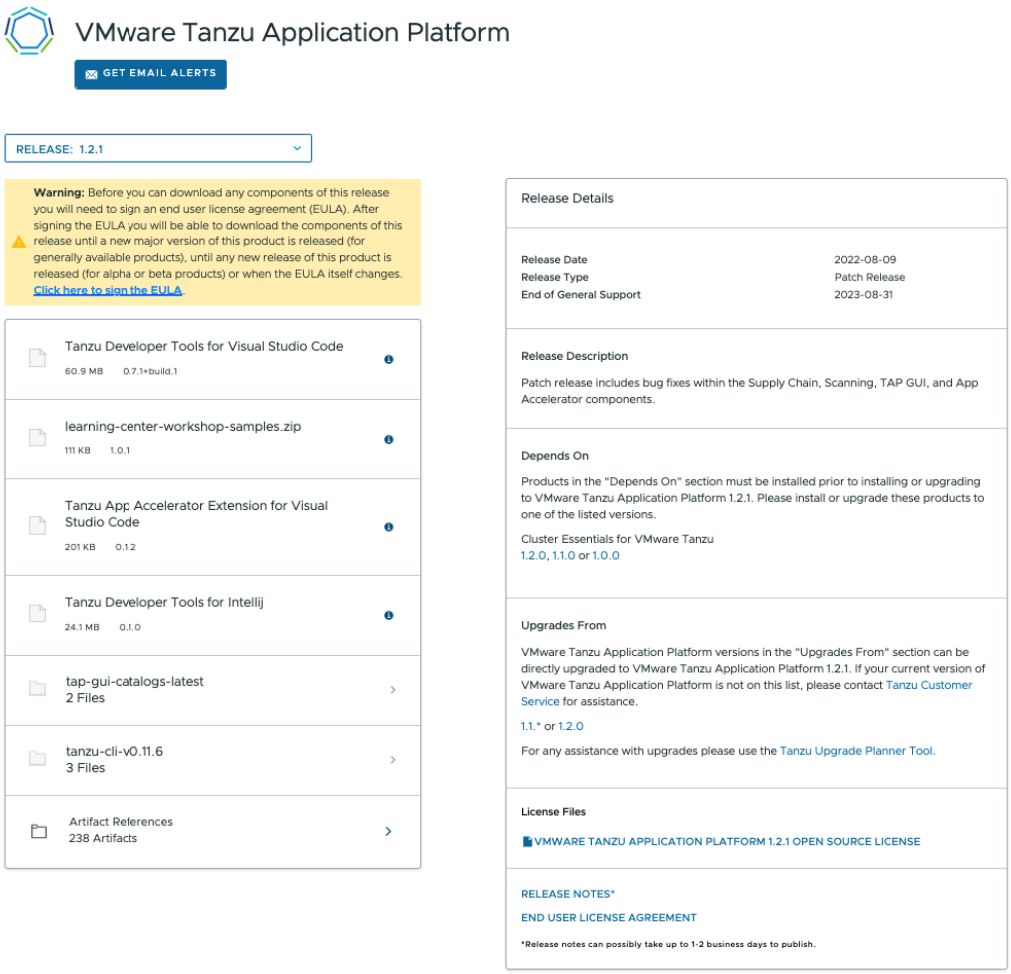
Before downloading and installing Tanzu Application Platform packages, you must accept the End User License Agreements (EULAs) as follows:

1. Sign in to VMware Tanzu Network.
2. Accept or confirm that you have accepted the EULAs for each of the following:
    - Tanzu Application Platform
    - Cluster Essentials for VMware Tanzu

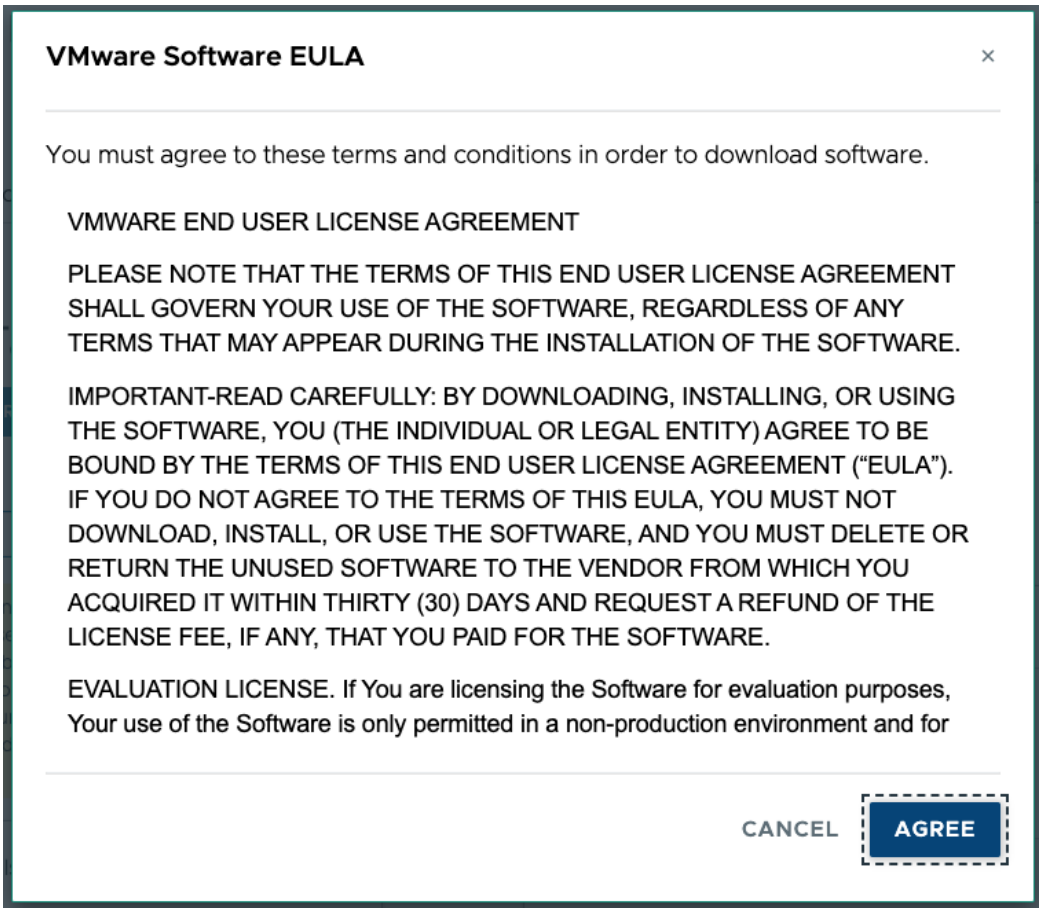
Example of accepting the Tanzu Application Platform EULA

To accept the Tanzu Application Platform EULA:

1. Go to Tanzu Application Platform.
2. Select the ***Click here to sign the EULA*** link in the yellow warning box under the release drop-down menu. If the yellow warning box is not visible, the EULA has already been accepted.



3. Select ***Agree*** in the bottom-right of the dialog box as seen in the following screenshot.



Set the Kubernetes cluster context

For information about the supported Kubernetes cluster providers and versions, see Kubernetes cluster requirements.

To set the Kubernetes cluster context:

1. List the existing contexts by running:

   ```
   kubectl config get-contexts
   ```

   For example:

   ```
   $ kubectl config get-contexts
   CURRENT   NAME                                CLUSTER           AUTHINFO
   NAMESPACE
           aks-repo-trial                      aks-repo-trial    clusterUser_aks-r
   g-01_aks-repo-trial
   *       aks-tap-cluster                     aks-tap-cluster   clusterUser_aks-r
   g-01_aks-tap-cluster
   ```

2. If you are managing multiple cluster contexts, set the context to the cluster that you want to use for the Tanzu Application Platform packages installation by running:

   ```
   kubectl config use-context CONTEXT
   ```

   Where `CONTEXT` is the cluster that you want to use. For example, `aks-tap-cluster`.

   For example:

```
$ kubectl config use-context aks-tap-cluster
Switched to context "aks-tap-cluster".
```

Install or update the Tanzu CLI and plug-ins

The Tanzu CLI and plug-ins enable you to install and use the Tanzu Application Platform functions and features.

Install the Tanzu CLI

The Tanzu CLI core v1.0.0 distributed with Tanzu Application Platform is forward and backward compatible with all supported releases of Tanzu Application Platform.

Run a single command to install the plug-in group version that matches the Tanzu Application Platform version on any target environment. For more information, see Install Tanzu CLI Plug-ins.

Use a package manager to install Tanzu CLI on Windows, Mac, or Linux OS. Alternatively, download and install manually from Tanzu Network, VMware Customer Connect, or GitHub.

Basic installation instructions are provided below. For more information including how to install the Tanzu CLI and CLI plug-ins in Internet-restricted environments, see the VMware Tanzu CLI documentation.

Note

To retain an existing installation of the Tanzu CLI, move the CLI binary from `/usr/local/bin/tanzu` or `C:\Program Files\tanzu` on Windows to a different location before following the steps below.

Install using a package manager
To install the Tanzu CLI using a package manager:

1. Follow the instructions for your package manager below. This installs the latest version of the CLI available in the package registry.
    - **Homebrew (MacOS):**

      ```
      brew update
      brew install vmware-tanzu/tanzu/tanzu-cli
      ```

    - **Chocolatey (Windows):**

      ```
      choco install tanzu-cli
      ```

      The `tanzu-cli` package is part of the main Chocolatey Community Repository. When a new `tanzu-cli` version is released, it might not be available immediately. If the above command fails, run:

      ```
      choco install tanzu-cli --version TANZU-CLI-VERSION
      ```

      Where `TANZU-CLI-VERSION` is the Tanzu CLI version you want to install.

      For example:

      ```
      choco install tanzu-cli --version 1.2.0
      ```

    - **APT (Debian or Ubuntu):**

```
sudo mkdir -p /etc/apt/keyrings/
sudo apt-get update
sudo apt-get install -y ca-certificates curl gpg
curl -fsSL https://packages.vmware.com/tools/keys/VMWARE-PACKAGING-GPG
-RSA-KEY.pub | sudo gpg --dearmor -o /etc/apt/keyrings/tanzu-archive-k
eyring.gpg
echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/tanzu-archive-keyrin
g.gpg] https://storage.googleapis.com/tanzu-cli-os-packages/apt tanzu-
cli-jessie main" | sudo tee /etc/apt/sources.list.d/tanzu.list
sudo apt-get update
sudo apt-get install -y tanzu-cli
```

- **YUM or DNF (RHEL):**

```
cat << EOF | sudo tee /etc/yum.repos.d/tanzu-cli.repo
[tanzu-cli]
name=Tanzu CLI
baseurl=https://storage.googleapis.com/tanzu-cli-os-packages/rpm/tanzu
-cli
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.vmware.com/tools/keys/VMWARE-PACKAGING-GPG-RSA
-KEY.pub
EOF

sudo yum install -y tanzu-cli # If you are using DNF, run sudo dnf ins
tall -y tanzu-cli.
```

2. Check that the correct version of the CLI is properly installed.

```
tanzu version
version: v1.2.0
...
```

Install from a binary release

To install the Tanzu CLI from a binary release:

1. Download the Tanzu CLI binary from one of the following locations:
   - **VMware Tanzu Network**
     1. Go to VMware Tanzu Network.
     2. Choose the 1.5.12 release from the Release dropdown menu.
     3. Click the tanzu-core-cli-binaries item from the result set.
     4. Download the Tanzu CLI binary for your operating system.
   - **VMware Customer Connect**
     1. Go to VMware Customer Connect.
     2. Download the Tanzu CLI binary for your operating system.
   - **GitHub**
     1. Go to Tanzu CLI release v1.2.0 on GitHub.
     2. Download the Tanzu CLI binary for your operating system, for example, `tanzu-cli-windows-amd64.tar.gz`.
2. Use an extraction tool to unpack the binary file:
   - **macOS:**

```
tar -xvf tanzu-cli-darwin-amd64.tar.gz
```

- **Linux:**

```
tar -xvf tanzu-cli-linux-amd64.tar.gz
```

- **Windows:**

  Use the Windows extractor tool to unzip `tanzu-cli-windows-amd64.zip`.

3. Make the CLI available to the system:
   - cd to the directory containing the extracted CLI binary
   - **macOS:**

     Install the binary to `/usr/local/bin`:

     ```
     install tanzu-cli-darwin_amd64 /usr/local/bin/tanzu
     ```

   - **Linux:**

     Install the binary to `/usr/local/bin`:

     ```
     sudo install tanzu-cli-linux_amd64 /usr/local/bin/tanzu
     ```

   - **Windows:**
     1. Create a new `Program Files\tanzu` folder.
     2. Copy the `tanzu-cli-windows_amd64.exe` file into the new `Program Files\tanzu` folder.
     3. Rename `tanzu-cli-windows_amd64.exe` to `tanzu.exe`.
     4. Right-click the `tanzu` folder, select **Properties** > **Security**, and make sure that your user account has the **Full Control** permission.
     5. Use Windows Search to search for `env`.
     6. Select **Edit the system environment variables** and click the **Environment Variables** button.
     7. Select the `Path` row under **System variables**, and click **Edit**.
     8. Click **New** to add a new row and enter the path to the Tanzu CLI. The path value must not include the `.exe` extension. For example, `C:\Program Files\tanzu`.
4. Check that the correct version of the CLI is properly installed:

   ```
   tanzu version
   version: v1.2.0
   ...
   ```

Install Tanzu CLI Plug-ins

There is a group of Tanzu CLI plug-ins which extend the Tanzu CLI Core with Tanzu Application Platform specific functionality. The plug-ins can be installed as a group with a single command.

Versioned releases of the Tanzu Application Platform specific plug-in group align to each supported Tanzu Application Platform version.

This makes it easy to switch between different versions of Tanzu Application Platforms environments. Use the following commands to search for, install, and verify Tanzu CLI plug-in groups.

List the versions of each plug-in group available across Tanzu

```
tanzu plugin group search --show-details
```

List the versions of the Tanzu Application Platform specific plug-in group

```
tanzu plugin group search --name vmware-tanzu/default --show-details
```

Install the version of the Tanzu Application Platform specific plug-in group matching your target environment

```
tanzu plugin install --group vmware-tap/default:v1.5.12
```

Verify the plugin group list against the plug-ins that were installed

```
tanzu plugin group get vmware-tap/default:v1.5.12
```

```
tanzu plugin list
```

For air-gapped installation, see the Installing the Tanzu CLI in Internet-Restricted Environments section of the Tanzu CLI documentation.

Next steps

For online installation:

- Deploy Cluster Essentials*
- Install the Tanzu Application Platform package and profiles

For air-gapped installation:

- Deploy Cluster Essentials*
- Install Tanzu Application Platform in an air-gapped environment

For GitOps (beta) installation:

- Deploy Cluster Essentials*
- Install Tanzu Application Platform through GitOps with ESO
- Install Tanzu Application Platform through Gitops with SOPS

** When you use a VMware Tanzu Kubernetes Grid cluster, you do not need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

Install Tanzu Application Platform (online)

To install Tanzu Application Platform (commonly known as TAP) on your Kubernetes clusters with internet access:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install the Tanzu Application Platform package and profiles
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

** When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform (online)

To install Tanzu Application Platform (commonly known as TAP) on your Kubernetes clusters with internet access:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install the Tanzu Application Platform package and profiles
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

** When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform package and profiles

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) packages from your Tanzu Application Platform package repository.

Before installing the packages, ensure you have:

- Completed the Prerequisites.
- Configured and verified the cluster.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following documentation for a registry to learn how to set it up:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

   ```
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
   export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
   export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own container registry.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`.
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.
   - `TARGET-REPOSITORY` is your target repository, a folder/repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

   VMware recommends using a JSON key file to authenticate with Google Container Registry. In this case, the value of `INSTALL_REGISTRY_USERNAME` is `_json_key` and the value of `INSTALL_REGISTRY_PASSWORD` is the content of the JSON key file. For more information about how to generate the JSON key file, see Google Container Registry documentation.

2. Install the Carvel tool imgpkg CLI.

To query for the available versions of Tanzu Application Platform on VMWare Tanzu Network Registry, run:

```
imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/tap-pac
kages | sort -V
```

3. Relocate the images with the `imgpkg` CLI by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-package
s:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-pac
kages
```

Add the Tanzu Application Platform package repository

Tanzu CLI packages are available on repositories. Adding the Tanzu Application Platform package repository makes Tanzu Application Platform and its packages available for installation.

Relocate images to a registry is strongly recommended but not required for installation. If you skip this step, you can use the following values to replace the corresponding variables:

- `INSTALL_REGISTRY_HOSTNAME` is `registry.tanzu.vmware.com`
- `INSTALL_REPO` is `tanzu-application-platform`
- `INSTALL_REGISTRY_USERNAME` and `INSTALL_REGISTRY_PASSWORD` are the credentials to the VMware Tanzu Network registry `registry.tanzu.vmware.com`
- `TAP_VERSION` is your Tanzu Application Platform version. For example, `1.5.12`

To add the Tanzu Application Platform package repository to your cluster:

1. Create a namespace called `tap-install` for deploying any component packages by running:

```
kubectl create ns tap-install
```

This namespace keeps the objects grouped together logically.

2. Create a registry secret by running:

```
tanzu secret registry add tap-registry \
  --username ${INSTALL_REGISTRY_USERNAME} --password ${INSTALL_REGISTRY_PASSWOR
D} \
  --server ${INSTALL_REGISTRY_HOSTNAME} \
  --export-to-all-namespaces --yes --namespace tap-install
```

3. Create a internal registry secret by running:

```
tanzu secret registry add registry-credentials \
    --server   ${INSTALL_REGISTRY_HOSTNAME} \
    --username ${INSTALL_REGISTRY_USERNAME} \
    --password ${INSTALL_REGISTRY_PASSWORD} \
    --namespace tap-install \
    --export-to-all-namespaces \
    --yes
```

4. Add the Tanzu Application Platform package repository to the cluster by running:

```
tanzu package repository add tanzu-tap-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages:$TAP_VERSION
\
  --namespace tap-install
```

5. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

For example:

```
$ tanzu package repository get tanzu-tap-repository --namespace tap-install
- Retrieving repository tap...
NAME:          tanzu-tap-repository
VERSION:       16253001
REPOSITORY:    tapmdc.azurecr.io/mdc/1.4.0/tap-packages
TAG:           1.5.12
STATUS:        Reconcile succeeded
REASON:
```

Note

The `VERSION` and `TAG` numbers differ from the earlier example if you are on Tanzu Application Platform v1.0.2 or earlier.

6. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com                   Application Accelerator
for VMware Tanzu                                    Used to create new projects a
nd configurations.
  api-portal.tanzu.vmware.com                         API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com                          API Auto Registration fo
r VMware Tanzu                                      A TAP component to automatica
lly register API exposing workloads as API entities

in TAP GUI.
  backend.appliveview.tanzu.vmware.com                Application Live View fo
r VMware Tanzu                                      App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com                       Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized

software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com          VMware Carbon Black for
Supply Chain Security Tools - Scan                  Default scan templates using
VMware Carbon Black
  cartographer.tanzu.vmware.com                       Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com                               Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com              Application Live View Co
nnector for VMware Tanzu                            App for discovering and regis
tering running apps
  controller.source.apps.tanzu.vmware.com             Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.
  conventions.appliveview.tanzu.vmware.com            Application Live View Co
```

```
nventions for VMware Tanzu                          Application Live View convent
ion server
  developer-conventions.tanzu.vmware.com              Tanzu App Platform Devel
oper Conventions                                    Developer Conventions
  eventing.tanzu.vmware.com                           Eventing
Eventing is an event-driven architecture platform based on Knative Eventing
  external-secrets.apps.tanzu.vmware.com              External Secrets Operato
r                                                   External Secrets Operator is
a Kubernetes operator that integrates external

secret management systems.
  fluxcd.source.controller.tanzu.vmware.com           Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts

acquisition from external sources such as Git, Helm repositories and S3 bucket
s.
  grype.scanning.apps.tanzu.vmware.com                Grype for Supply Chain S
ecurity Tools - Scan                                Default scan templates using
Anchore Grype
  learningcenter.tanzu.vmware.com                     Learning Center for Tanz
u Application Platform                              Guided technical workshops
  metadata-store.apps.tanzu.vmware.com                Supply Chain Security To
ols - Store                                         Post SBoMs and query for imag
e, package, and vulnerability metadata.
  namespace-provisioner.apps.tanzu.vmware.com         Namespace Provisioner
Automatic Provisioning of Developer Namespaces.
  ootb-delivery-basic.tanzu.vmware.com                Tanzu App Platform Out o
f The Box Delivery Basic                            Out of The Box Delivery Basi
c.
  ootb-supply-chain-basic.tanzu.vmware.com            Tanzu App Platform Out o
f The Box Supply Chain Basic                        Out of The Box Supply Chain B
asic.
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning  Out of The Box Supply Chain w
ith Testing and Scanning.
  ootb-supply-chain-testing.tanzu.vmware.com          Tanzu App Platform Out o
f The Box Supply Chain with Testing                 Out of The Box Supply Chain w
ith Testing.
  ootb-templates.tanzu.vmware.com                     Tanzu App Platform Out o
f The Box Templates                                 Out of The Box Templates.
  policy.apps.tanzu.vmware.com                        Supply Chain Security To
ols - Policy Controller                             Policy Controller enables def
ining of a policy to restrict unsigned container

images.
  scanning.apps.tanzu.vmware.com                      Supply Chain Security To
ols - Scan                                          Scan for vulnerabilities and
enforce policies directly within Kubernetes native

Supply Chains.
  service-bindings.labs.vmware.com                    Service Bindings for Kub
ernetes                                             Service Bindings for Kubernet
es implements the Service Binding Specification.
  services-toolkit.tanzu.vmware.com                   Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and

connectivity of Service Resources (databases, message queues, DNS records,

etc.).
  snyk.scanning.apps.tanzu.vmware.com                 Snyk for Supply Chain Se
curity Tools - Scan                                 Default scan templates using
Snyk
  spring-boot-conventions.tanzu.vmware.com            Tanzu Spring Boot Conven
tions Server                                        Default Spring Boot conventio
n server.
  sso.apps.tanzu.vmware.com                           AppSSO
```

```
Application Single Sign-On for Tanzu
  tap-auth.tanzu.vmware.com                              Default roles for Tanzu
Application Platform                                   Default roles for Tanzu Appli
cation Platform
  tap-gui.tanzu.vmware.com                               Tanzu Application Platfo
rm GUI                                                 web app graphical user interf
ace for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                         Telemetry Collector for
Tanzu Application Platform                             Tanzu Application Plaform Tel
emetry
  tap.tanzu.vmware.com                                   Tanzu Application Platfo
rm                                                     Package to install a set of T
AP components to get you started based on your use

case.
  tekton.tanzu.vmware.com                                Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.
  workshops.learningcenter.tanzu.vmware.com              Workshop Building Tutori
al                                                     Workshop Building Tutorial
```

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings. This is done by using the package manager installed by Tanzu Cluster Essentials.

For more information about profiles, see Components and installation profiles.

To prepare to install a profile:

1. List version information for the package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. Create a `tap-values.yaml` file by using the Full Profile sample in the following section as a guide. These samples have the minimum configuration required to deploy Tanzu Application Platform. The sample values file contains the necessary defaults for:

   - The meta-package, or parent Tanzu Application Platform package.
   - Subordinate packages, or individual child packages.

   Keep the values file for future configuration use.

   Note

   `tap-values.yaml` is set as a Kubernetes secret, which provides secure means to read credentials for Tanzu Application Platform components.

3. View possible configuration settings for your package

Full profile

The following is the YAML file sample for the full-profile. The `profile:` field takes `full` as the default value, but you can also set it to `iterate`, `build`, `run` or `view`. Refer to Install multicluster Tanzu Application Platform profiles for more information.

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  ingress_issuer: # Optional, can denote a cert-manager.io/v1/ClusterIssuer of your ch
oice. Defaults to "tap-ingress-selfsigned".
```

```
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
    secret:
      name: "KP-DEFAULT-REPO-SECRET"
      namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"

  kubernetes_distribution: "K8S-DISTRO" # Only required if the distribution is OpenShi
ft and must be used with the following kubernetes_version key.

  kubernetes_version: "K8S-VERSION" # Required regardless of distribution when Kuberne
tes version is 1.25 or later.

  ca_cert_data: | # To be passed if using custom certificates.
      -----BEGIN CERTIFICATE-----
      MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
      -----END CERTIFICATE-----

ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

#The above keys are minimum numbers of entries needed in tap-values.yaml to get a func
tioning TAP Full profile installation.

#Below are the keys which may have default values set, but can be overridden.

profile: full # Can take iterate, build, run, view.

supply_chain: basic # Can take testing, testing_scanning.

ootb_supply_chain_basic: # Based on supply_chain set above, can be changed to ootb_sup
ply_chain_testing, ootb_supply_chain_testing_scanning.
  registry:
    server: "SERVER-NAME" # Takes the value from the shared section by default, but ca
n be overridden by setting a different value.
    repository: "REPO-NAME" # Takes the value from the shared section by default, but
can be overridden by setting a different value.
  gitops:
    ssh_secret: "SSH-SECRET-KEY" # Takes "" as value by default; but can be overridden
by setting a different value.

contour:
  envoy:
    service:
      type: LoadBalancer # This is set by default, but can be overridden by setting a
different value.

buildservice:
  # Takes the value from the shared section by default, but can be overridden by setti
ng a different value.
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section above by def
ault, but can be overridden by setting a different value.
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"

tap_gui:
  metadataStoreAutoconfiguration: true # Creates a service account, the Kubernetes con
trol plane token and the requisite app_config block to enable communications between T
anzu Application Platform GUI and SCST - Store.
  app_config:
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml

metadata_store:
```

```
  ns_for_export_app_cert: "MY-DEV-NAMESPACE" # Verify this namespace is available with
in your cluster before initiating the Tanzu Application Platform installation.
  app_service_type: ClusterIP # Defaults to LoadBalancer. If shared.ingress_domain is
set earlier, this must be set to ClusterIP.

scanning:
  metadataStore:
    url: "" # Configuration is moved, so set this string to empty.

grype:
  namespace: "MY-DEV-NAMESPACE" # Verify this namespace is available within your clust
er before initiating the Tanzu Application Platform installation.
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
  # In a single cluster, the connection between the scanning pod and the metadata stor
e happens inside the cluster and does not pass through ingress. This is automatically
configured, you do not need to provide an ingress connection to the store.

policy:
  tuf_enabled: false # By default, TUF initialization and keyless verification are dea
ctivated.
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating the Tanzu Application Platform usage reports.
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address. It is not required to know the External IP address or set up the DNS record while installing. Installing the Tanzu Application Platform package creates the `tanzu-shared-ingress` and its External IP address. You can create the DNS record after completing the installation.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
    - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
    - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
    - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-SECRET` is the secret with user credentials that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
    - You can create a secret configured with a valid registry credential with a name and namespace of your choice. For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
    - You must create the secret before the installation. For example, you can use the `registry-credentials` secret created earlier.
- `KP-DEFAULT-REPO-SECRET-NAMESPACE` is the namespace where `KP-DEFAULT-REPO-SECRET` is created.
    - You must create the namespace before the installation. For example, you can use the `tap-install` namespace created earlier.
- `K8S-DISTRO` (optional) is the type of Kubernetes infrastructure in use. It is only required if the distribution is OpenShift and must be used in coordination with `kubernetes_version`. Supported value: `openshift`.
- `K8S-VERSION` (optional) is the Kubernetes version in use. You can use it independently or in coordination with `kubernetes_distribution`. For example, `1.24.x`, where `x` is the

Kubernetes patch version.

- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. If this key is passed through the shared section earlier and AWS ECR registry is used, you must ensure that the `SERVER-NAME/REPO-NAME/buildservice` and `SERVER-NAME/REPO-NAME/workloads` exist. AWS ECR expects the paths to be pre-created. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. This field is only required if you use a private repository, otherwise, leave it empty. See Git authentication for more information.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the SCST - Store CA certificate to all namespaces. To install Grype in multiple namespaces, use a namespace provisioner. For more information, see Namespace Provisioner.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports. See Locating the Entitlement Account number for new orders for more information about identifying the Entitlement Account Number.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

If you use AWS, the default settings creates a classic LoadBalancer. To use the Network LoadBalancer instead of the classic LoadBalancer for ingress, add the following to your `tap-values.yaml`:

```
contour:
  infrastructure_provider: aws
  envoy:
    service:
      aws:
        LBType: nlb
```

CEIP policy disclosure

Tanzu Application Platform is part of VMware's CEIP program where data is collected to help improve the customer experience. By setting `ceip_policy_disclosed` to `true` (not a string), you acknowledge the program is disclosed to you and you are aware data collection is happening. This field must be set for the installation to be completed.

See Opt out of telemetry collection for more information.

(Optional) Additional Build Service configurations

The following tasks are optional during the Tanzu Application Platform installation process:

- (Optional) Configure your profile with full dependencies
- (Optional) Configure your profile with the Jammy stack only

(Optional) Configure your profile with full dependencies

When you install a profile that includes Tanzu Build Service, Tanzu Application Platform is installed with the `lite` set of dependencies. These dependencies consist of buildpacks and stacks required for application builds.

The `lite` set of dependencies do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section by default,
but can be overridden by setting a different value.
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  exclude_dependencies: true
```

After configuring `full` dependencies, you must install the dependencies after you have finished installing your Tanzu Application Platform package. See Install the full dependencies package for more information.

(Optional) Configure your profile with the Jammy stack only

Tanzu Application Platform v1.5.0 supports building applications with both the Ubuntu v22.04 (Jammy) and v18.04 (Bionic) stack. For more information, see Bionic and Jammy stacks.

To install Tanzu Application Platform with Jammy as the only available stack, include the `stack_configuration: jammy-only` field under the `buildservice:` section in `tap-values.yaml`.

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v $TAP_VERSION --values-file
   tap-values.yaml -n tap-install
   ```

2. Verify the package install by running:

   ```
   tanzu package installed get tap -n tap-install
   ```

This can take 5-10 minutes because it installs several packages on your cluster.

3. Verify that the necessary packages in the profile are installed by running:

```
tanzu package installed list -A
```

4. If you configured `full` dependencies in your `tap-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

Important

After installing the full profile on your cluster, you must set up developer namespaces. Otherwise, creating a workload, a Knative service or other Tanzu Application Platform packages fails. For more information, see Set up developer namespaces to use your installed packages.

You can run the following command after reconfiguring the profile to reinstall the Tanzu Application Platform:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v $TAP_VERSION  --values-f
ile tap-values.yaml -n tap-install
```

Install the full dependencies package

If you configured `full` dependencies in your `tap-values.yaml` file in Configure your profile with full dependencies earlier, you must install the `full` dependencies package.

For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To install the `full` dependencies package:

1. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
...
  exclude_dependencies: true
...
```

2. Get the latest version of the `buildservice` package by running:

```
tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
all
```

3. Relocate the Tanzu Build Service full dependencies package repository by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
ps-package-repo:VERSION \
  --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tbs-full-deps
```

Where `VERSION` is the version of the `buildservice` package you retrieved in the previous step.

4. Add the Tanzu Build Service full dependencies package repository by running:

```
tanzu package repository add tbs-full-deps-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tbs-full-deps:VERSION \
```

```
  --namespace tap-install
```

Where `VERSION` is the version of the `buildservice` package you retrieved earlier.

5. Install the full dependencies package by running:

```
tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
N -n tap-install
```

Where `VERSION` is the version of the `buildservice` package you retrieved earlier.

Access Tanzu Application Platform GUI

To access Tanzu Application Platform GUI, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Application Platform GUI, see Access Tanzu Application Platform GUI.

You're now ready to start using Tanzu Application Platform GUI. Proceed to the Getting Started topic or the Tanzu Application Platform GUI - Catalog Operations topic.

Exclude packages from a Tanzu Application Platform profile

To exclude packages from a Tanzu Application Platform profile:

1. Find the full subordinate (child) package name:

```
tanzu package available list --namespace tap-install
```

2. Update your `tap-values` file with a section listing the exclusions:

```
profile: PROFILE-VALUE
excluded_packages:
  - tap-gui.tanzu.vmware.com
  - service-bindings.lab.vmware.com
```

Important

If you exclude a package after performing a profile installation including that package, you cannot see the accurate package states immediately after running `tap package installed list -n tap-install`. Also, you can break package dependencies by removing a package. Allow 20 minutes to verify that all packages have reconciled correctly while troubleshooting.

Next steps

- (Optional) Install individual packages
- Set up developer namespaces to use your installed packages
- Replace the default ingress issuer

View possible configuration settings for your package

To view possible configuration settings for a package, run:

```
tanzu package available get tap.tanzu.vmware.com/$TAP_VERSION --values-schema --namesp
ace tap-install
```

Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNRS-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```
profile: full

# Shared configurations go under the shared key.
shared:
  ingress_domain: tap.example.com

# ...

# For example, CNRs specific values go under its name.
cnrs:
  provider: local

# For example, App Accelerator specific values go under its name.
accelerator:
  server:
    service_type: "ClusterIP"
```

Shared Keys define values that configure multiple packages. These keys are defined under the `shared` Top-level Key, as summarized in the following table:

Shared Key	Description	Optional
`ca_cert_data`	PEM-encoded certificate data to trust TLS connections with a private CA. This shared key is used by `convention_controller`, `scanning` and `source_controller`	Yes
`ingress_domain`	Domain name to be used in service routes and host names for instances of Tanzu Application Platform components.	Yes
`ingress_issuer`	A `cert-manager.io/v1/ClusterIssuer` for issuing TLS certificates to Tanzu Application Platform components. Default value: `tap-ingress-selfsigned`	Yes
`kubernetes_distribution`	Type of Kubernetes infrastructure being used. You can use this shared key in coordination with the `kubernetes_version` key. Supported value: `openshift`.	Yes
`kubernetes_version`	Kubernetes version. You can use this shared key independently or in coordination with the `kubernetes_distribution` key. Supported value: `1.24.x`, where `x` stands for the Kubernetes patch version.	Yes
`image_registry.project_path`	Project path in the container image registry server used for builder and application images.	Yes
`image_registry.username`	User name for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`image_registry.password`	Password for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes

Shared Key	Description	Optional
`secret.name`	Secret name for the container image registry credentials of type `kubernetes.io/dockerconfigjson`. Mutually exclusive with `shared.image_registry.username/password`	Yes
`secret.namespace`	Secret namespace for the container image registry credentials. Mutually exclusive with `shared.image_registry.username/password`	Yes
`activateAppLiveViewSecureAccessControl`	Enable secure access connection between Application Live View components.	Yes

The following table summarizes the top-level keys used for package-specific configuration within your `tap-values.yaml`.

Package	Top-level Key
See table above.	`shared`
API Auto Registration	`api_auto_registration`
API portal	`api_portal`
Application Accelerator	`accelerator`
Application Live View	`appliveview`
Application Live View connector	`appliveview_connector`
Application Live View conventions	`appliveview-conventions`
Cartographer	`cartographer`
Cloud Native Runtimes	`cnrs`
Source Controller	`source_controller`
Supply Chain	`supply_chain`
Supply Chain Basic	`ootb_supply_chain_basic`
Supply Chain Testing	`ootb_supply_chain_testing`
Supply Chain Testing Scanning	`ootb_supply_chain_testing_scanning`
Supply Chain Security Tools - Scan	`scanning`
Supply Chain Security Tools - Scan (Grype Scanner)	`grype`
Supply Chain Security Tools - Store	`metadata_store`
Build Service	`buildservice`
Tanzu Application Platform GUI	`tap_gui`
Learning Center	`learningcenter`

For information about package-specific configuration, see Install individual packages.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before

installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service
- Install Application Live View
- Install Application Single Sign-On
- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform
- Install Developer Conventions
- Install Eventing
- Install Flux CD Source Controller
- Install Learning Center for Tanzu Application Platform
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit
- Install Source Controller
- Install Spring Boot conventions
- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Application Platform GUI
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package installed list --namespace tap-install
   \ Retrieving installed packages...
   NAME                    PACKAGE-NAME                                    PAC
   KAGE-VERSION  STATUS
   api-portal              api-portal.tanzu.vmware.com                     1.
   0.3           Reconcile succeeded
   app-accelerator         accelerator.apps.tanzu.vmware.com               1.
   0.0           Reconcile succeeded
   app-live-view           appliveview.tanzu.vmware.com                    1.
   0.2           Reconcile succeeded
   appliveview-conventions build.appliveview.tanzu.vmware.com              1.
   0.2           Reconcile succeeded
   cartographer            cartographer.tanzu.vmware.com                   0.
   1.0           Reconcile succeeded
   cloud-native-runtimes   cnrs.tanzu.vmware.com                           1.
   0.3           Reconcile succeeded
   convention-controller   controller.conventions.apps.tanzu.vmware.com    0.
   7.0           Reconcile succeeded
   developer-conventions   developer-conventions.tanzu.vmware.com          0.
   3.0-build.1   Reconcile succeeded
   grype-scanner           grype.scanning.apps.tanzu.vmware.com            1.
   0.0           Reconcile succeeded
   image-policy-webhook    image-policy-webhook.signing.apps.tanzu.vmware.com 1.
   1.2           Reconcile succeeded
   metadata-store          metadata-store.apps.tanzu.vmware.com            1.
   0.2           Reconcile succeeded
   ootb-supply-chain-basic ootb-supply-chain-basic.tanzu.vmware.com        0.
   5.1           Reconcile succeeded
   ootb-templates          ootb-templates.tanzu.vmware.com                 0.
   5.1           Reconcile succeeded
   scan-controller         scanning.apps.tanzu.vmware.com                  1.
   0.0           Reconcile succeeded
   service-bindings        service-bindings.labs.vmware.com                0.
   5.0           Reconcile succeeded
   services-toolkit        services-toolkit.tanzu.vmware.com               0.
   8.0           Reconcile succeeded
   source-controller       controller.source.apps.tanzu.vmware.com         0.
   2.0           Reconcile succeeded
   sso4k8s-install         sso.apps.tanzu.vmware.com                       1.
   0.0-beta.2-31 Reconcile succeeded
   tap-gui                 tap-gui.tanzu.vmware.com                        0.
   3.0-rc.4      Reconcile succeeded
   tekton-pipelines        tekton.tanzu.vmware.com                         0.3
   0.0          Reconcile succeeded
   tbs                     buildservice.tanzu.vmware.com                   1.
   5.0           Reconcile succeeded
   ```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

For details about how to automatically set up your developer namespaces, see Provision developer namespaces in Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see Legacy namespace setup.

Next steps

- Install Tanzu Developer Tools for your VS Code

Provision namespaces manually

This topic tells you how to use Namespace Provisioner to provision namespaces manually in Tanzu Application Platform (commonly known as TAP).

Using Namespace Provisioner is the recommended best practice for setting up developer namespaces on Tanzu Application Platform.

To provision namespaces manually, complete the following steps:

1. Enable single user access.
2. (Optional) Enable additional users with Kubernetes RBAC.

Enable single user access

1. To add read/write registry credentials to the developer namespace, run the following command:

   ```
   tanzu secret registry add registry-credentials --server REGISTRY-SERVER --usern
   ame REGISTRY-USERNAME --password REGISTRY-PASSWORD --namespace YOUR-NAMESPACE
   ```

   Where:

   - `YOUR-NAMESPACE` is the name you give to the developer namespace. For example, use `default` for the default namespace.
   - `REGISTRY-SERVER` is the URL of the registry. You can use the same registry server as in `ootb_supply_chain_basic` - `registry` - `server`. For more information, see Install Tanzu Application Platform package and profiles.
     - For Docker Hub, the value is `https://index.docker.io/v1/`. It must have the leading `https://`, the `v1` path, and the trailing `/`.
     - For Google Container Registry (GCR), the value is `gcr.io`.
   - `REGISTRY-PASSWORD` is the password of the registry.
     - For GCR or Google Artifact Registry, this must be the concatenated version of the JSON key. For example: `"$(cat ~/gcp-key.json)"`

   If you observe the following issue:

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x128 pc=0x2bcce00]
```

Use kubectl to create the secret instead:

```
kubectl create secret docker-registry registry-credentials --docker-server=REGI
STRY-SERVER --docker-username=REGISTRY-USERNAME --docker-password=REGISTRY-PASS
WORD -n YOUR-NAMESPACE
```

Note

This step is not required if you install Tanzu Application Platform on AWS with EKS and use IAM Roles for Kubernetes Service Accounts instead of secrets. You can specify the Role Amazon Resource Name (ARN) in the next step.

2. Run the following to add secrets, a service account to execute the supply chain, and RBAC rules to authorize the service account to the developer namespace:

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: e30K
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-deliverable
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: deliverable
subjects:
  - kind: ServiceAccount
    name: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-workload
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: workload
subjects:
```

```
  - kind: ServiceAccount
    name: default
EOF
```

Note

If you install Tanzu Application Platform on AWS with EKS and use IAM Roles for Kubernetes Service Accounts, you must annotate the ARN of the IAM Role and remove the `registry-credentials` secret. Your service account entry then looks like the following:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
  annotations:
    eks.amazonaws.com/role-arn: <Role ARN>
imagePullSecrets:
  - name: tap-registry
```

Enable additional users with Kubernetes RBAC

Follow these steps to enable additional users in your namespace by using Kubernetes RBAC:

1. (Optional) Before you begin, ensure that you have enabled single user access. If you've set up your developer namespace using Namespace Provisioner, you can skip this step.
2. Choose either of the following options to give developers namespace-level access and view access to the appropriate cluster-level resources:
    - **Option 1:** Use the Tanzu Application Platform RBAC CLI plug-in (beta).

      To use the `tanzu rbac` plug-in to grant `app-viewer` and `app-editor` roles to an identity provider group, run:

      ```
      tanzu rbac binding add -g GROUP-FOR-APP-VIEWER -n YOUR-NAMESPACE -r app-v
      iewer
      tanzu rbac binding add -g GROUP-FOR-APP-EDITOR -n YOUR-NAMESPACE -r app-e
      ditor
      ```

      Where:

      - `YOUR-NAMESPACE` is the name you give to the developer namespace.
      - `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
      - `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

      For more information about `tanzu rbac`, see Bind a user or group to a default role

      VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

      Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set

up Azure Active Directory (Azure AD) with your cluster, see Integrate Azure Active Directory.

- **Option 2:** Use the native Kubernetes YAML.

  Run the following to apply the RBAC policy:

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-permit-app-viewer
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-viewer
subjects:
  - kind: Group
    name: GROUP-FOR-APP-VIEWER
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: YOUR-NAMESPACE-permit-app-viewer
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-viewer-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-VIEWER
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: YOUR-NAMESPACE-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
EOF
```

  Where:

  - `YOUR-NAMESPACE` is the name you give to the developer namespace.

- `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
- `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster.

Rather than granting roles directly to individuals, VMware recommends using your identity provider's user groups system to grant access to a group of developers.

For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrate Azure Active Directory.

3. (Optional) Log in as a non-admin user, such as a developer, to see the effects of RBAC after the role bindings are applied.

Additional configuration for testing and scanning

If you plan to install Out of the Box Supply Chains with Testing and Scanning, see Developer Namespace.

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

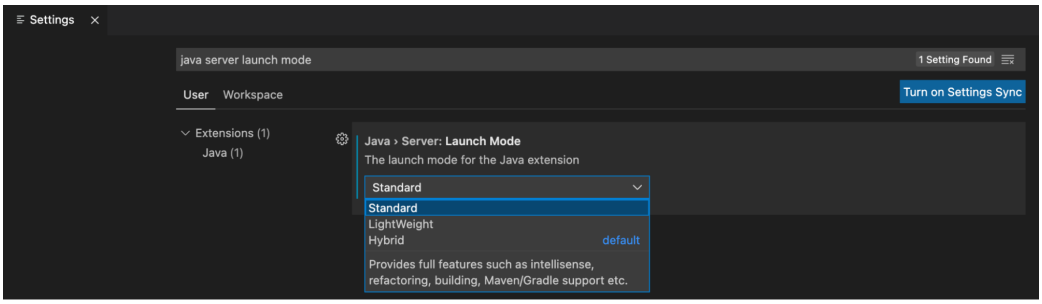
Install

To install the extension:

1. Sign in to VMware Tanzu Network and download Tanzu Developer Tools for Visual Studio Code.
2. Open VS Code.

3. Press cmd+shift+P to open the Command Palette and run `Extensions: Install from VSIX...`.

4. Select the extension file **tanzu-vscode-extension.vsix**.
5. If you do not have the following extensions, and they do not automatically install, install them from VS Code Marketplace:
    - Debugger for Java
    - Language Support for Java(™) by Red Hat
    - YAML
6. Ensure Language Support for Java is running in Standard Mode. You can configure it in the **Settings** menu by going to **Code** > **Preferences** > **Settings** under **Java > Server: Launch Mode**.



When the JDK and Language Support for Java are configured correctly, you see that the integrated development environment creates a directory target where the code is compiled.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: For more information, see Integrating Live Hover by using Spring Boot Tools. Reload VS Code for this change to take effect.
    - **Source Image**: (Required) The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name.
    - **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
    - **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in `kubeconfig` is the default.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and select **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Install Tanzu Application Platform (offline)

To install Tanzu Application Platform (commonly known as TAP) on your Kubernetes clusters in an air-gapped environment:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install Tanzu Application Platform in an air-gapped environment
5.	Install Tanzu Build Service full dependencies.	Install the Tanzu Build Service dependencies
6.	Configure custom certificate authorities for Tanzu Application Platform GUI.	Configure custom certificate authorities for Tanzu Application Platform GUI
7.	Add the certificate for the private Git repository in the Accelerator system namespace.	Configure Application Accelerator
8.	Apply patch to Grype.	Use Grype in offline and air-gapped environments
9.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages

* *When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your air-gapped cluster, you can start creating workloads that run in your air-gapped containers.

For more information about the Namespace Provisioner mode, see Work with Git repositories in air-gapped environments with Namespace Provisioner.

For more information about the manual mode, see Deploy an air-gapped workload.

Install Tanzu Application Platform (offline)

To install Tanzu Application Platform (commonly known as TAP) on your Kubernetes clusters in an air-gapped environment:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites

Step	Task	Link
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install Tanzu Application Platform in an air-gapped environment
5.	Install Tanzu Build Service full dependencies.	Install the Tanzu Build Service dependencies
6.	Configure custom certificate authorities for Tanzu Application Platform GUI.	Configure custom certificate authorities for Tanzu Application Platform GUI
7.	Add the certificate for the private Git repository in the Accelerator system namespace.	Configure Application Accelerator
8.	Apply patch to Grype.	Use Grype in offline and air-gapped environments
9.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages

* *When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your air-gapped cluster, you can start creating workloads that run in your air-gapped containers.

For more information about the Namespace Provisioner mode, see Work with Git repositories in air-gapped environments with Namespace Provisioner.

For more information about the manual mode, see Deploy an air-gapped workload.

Install Tanzu Application Platform in your air-gapped environment

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) on your Kubernetes cluster and registry that are air-gapped from external traffic.

Before installing the packages, ensure that you have completed the following tasks:

- Review the Prerequisites to ensure that you have set up everything required before beginning the installation.
- Accept Tanzu Application Platform EULA and install Tanzu CLI.
- Deploy Cluster Essentials. This step is optional if you are using VMware Tanzu Kubernetes Grid cluster.

Relocate images to a registry

To relocate images from the VMware Tanzu Network registry to your air-gapped registry:

1. Set up environment variables for installation use by running:

```
export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
```

```
export TAP_VERSION=VERSION-NUMBER
export REGISTRY_CA_PATH=PATH-TO-CA
export TO_REPO=MY-REPO
```

Where:

- `MY-REGISTRY` is your air-gapped container registry.
- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
- `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`
- `MY-REPO` is your repository in the air-gapped container image registry. Examples:
  - Harbor has the form `MY-REGISTRY/REPO-NAME/tap-packages`.
  - Docker Hub has the form `MY-REGISTRY/tap-packages`.
  - Google Cloud Registry has the form `MY-REGISTRY/MY-PROJECT/REPO-NAME/tap-packages`.

2. Copy the images into a `.tar` file from the VMware Tanzu Network onto an external storage device with the Carvel tool imgpkg by running:

```
imgpkg copy \
  -b registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:$TAP_VER
SION \
  --to-tar tap-packages-$TAP_VERSION.tar \
  --include-non-distributable-layers
```

3. Relocate the images with the Carvel tool imgpkg by running:

```
imgpkg copy \
  --tar tap-packages-$TAP_VERSION.tar \
  --to-repo $TO_REPO \
  --include-non-distributable-layers \
  --registry-ca-cert-path $REGISTRY_CA_PATH
```

4. Create a namespace called `tap-install` for deploying any component packages by running:

```
kubectl create ns tap-install
```

This namespace keeps the objects grouped together logically.

5. Create a registry secret by running:

```
tanzu secret registry add tap-registry \
    --server   $IMGPKG_REGISTRY_HOSTNAME_1 \
    --username $IMGPKG_REGISTRY_USERNAME_1 \
    --password $IMGPKG_REGISTRY_PASSWORD_1 \
    --namespace tap-install \
    --export-to-all-namespaces \
    --yes
```

6. Create a internal registry secret by running:

```
tanzu secret registry add registry-credentials \
    --server   $IMGPKG_REGISTRY_HOSTNAME_1 \
    --username $IMGPKG_REGISTRY_USERNAME_1 \
```

```
    --password $IMGPKG_REGISTRY_PASSWORD_1 \
    --namespace tap-install \
    --export-to-all-namespaces \
    --yes
```

7. Add the Tanzu Application Platform package repository to the cluster by running:

```
tanzu package repository add tanzu-tap-repository \
  --url $IMGPKG_REGISTRY_HOSTNAME_1/tap-packages:$TAP_VERSION \
  --namespace tap-install
```

   Where `$TAP_VERSION` is the Tanzu Application Platform version environment variable you defined earlier.

8. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

   Note

   The `VERSION` and `TAG` numbers differ from the earlier example if you are on Tanzu Application Platform v1.0.2 or earlier.

9. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

   For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com                   Application Accelerator
for VMware Tanzu                                    Used to create new projects a
nd configurations.
  api-portal.tanzu.vmware.com                         API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com                          API Auto Registration fo
r VMware Tanzu                                      A TAP component to automatica
lly register API exposing workloads as API entities

in TAP GUI.
  backend.appliveview.tanzu.vmware.com                Application Live View fo
r VMware Tanzu                                      App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com                       Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized

software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com          VMware Carbon Black for
Supply Chain Security Tools - Scan                  Default scan templates using
VMware Carbon Black
  cartographer.tanzu.vmware.com                       Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com                               Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com              Application Live View Co
nnector for VMware Tanzu                            App for discovering and regis
tering running apps
```

```
  controller.source.apps.tanzu.vmware.com              Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.
  conventions.appliveview.tanzu.vmware.com             Application Live View Co
nventions for VMware Tanzu                           Application Live View convent
ion server
  developer-conventions.tanzu.vmware.com               Tanzu App Platform Devel
oper Conventions                                     Developer Conventions
  eventing.tanzu.vmware.com                                 Eventing
Eventing is an event-driven architecture platform based on Knative Eventing
  external-secrets.apps.tanzu.vmware.com               External Secrets Operato
r                                                    External Secrets Operator is
a Kubernetes operator that integrates external

secret management systems.
  fluxcd.source.controller.tanzu.vmware.com            Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts

acquisition from external sources such as Git, Helm repositories and S3 bucket
s.
  grype.scanning.apps.tanzu.vmware.com                 Grype for Supply Chain S
ecurity Tools - Scan                                 Default scan templates using
Anchore Grype
  learningcenter.tanzu.vmware.com                           Learning Center for Tanz
u Application Platform                               Guided technical workshops
  metadata-store.apps.tanzu.vmware.com                      Supply Chain Security To
ols - Store                                          Post SBoMs and query for imag
e, package, and vulnerability metadata.
  namespace-provisioner.apps.tanzu.vmware.com          Namespace Provisioner
Automatic Provisioning of Developer Namespaces.
  ootb-delivery-basic.tanzu.vmware.com                      Tanzu App Platform Out o
f The Box Delivery Basic                             Out of The Box Delivery Basi
c.
  ootb-supply-chain-basic.tanzu.vmware.com                  Tanzu App Platform Out o
f The Box Supply Chain Basic                         Out of The Box Supply Chain B
asic.
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning  Out of The Box Supply Chain w
ith Testing and Scanning.
  ootb-supply-chain-testing.tanzu.vmware.com                Tanzu App Platform Out o
f The Box Supply Chain with Testing                  Out of The Box Supply Chain w
ith Testing.
  ootb-templates.tanzu.vmware.com                           Tanzu App Platform Out o
f The Box Templates                                  Out of The Box Templates.
  policy.apps.tanzu.vmware.com                              Supply Chain Security To
ols - Policy Controller                              Policy Controller enables def
ining of a policy to restrict unsigned container

images.
  scanning.apps.tanzu.vmware.com                            Supply Chain Security To
ols - Scan                                           Scan for vulnerabilities and
enforce policies directly within Kubernetes native

Supply Chains.
  service-bindings.labs.vmware.com                          Service Bindings for Kub
ernetes                                              Service Bindings for Kubernet
es implements the Service Binding Specification.
  services-toolkit.tanzu.vmware.com                         Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and

connectivity of Service Resources (databases, message queues, DNS records,

etc.).
  snyk.scanning.apps.tanzu.vmware.com                       Snyk for Supply Chain Se
curity Tools - Scan                                  Default scan templates using
Snyk
  spring-boot-conventions.tanzu.vmware.com             Tanzu Spring Boot Conven
```

```
tions Server                                        Default Spring Boot conventio
n server.
  sso.apps.tanzu.vmware.com                              AppSSO
Application Single Sign-On for Tanzu
  tap-auth.tanzu.vmware.com                              Default roles for Tanzu
Application Platform                                Default roles for Tanzu Appli
cation Platform
  tap-gui.tanzu.vmware.com                               Tanzu Application Platfo
rm GUI                                              web app graphical user interf
ace for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                         Telemetry Collector for
Tanzu Application Platform                          Tanzu Application Platform Te
lemetry
  tap.tanzu.vmware.com                                   Tanzu Application Platfo
rm                                                  Package to install a set of T
AP components to get you started based on your use

case.
  tekton.tanzu.vmware.com                                Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.
  workshops.learningcenter.tanzu.vmware.com              Workshop Building Tutori
al                                                  Workshop Building Tutorial
```

Prepare Sigstore Stack for air-gapped policy controller

Important

This section only applies if the target environment requires support for keyless authorities in `ClusterImagePolicy`. You must set the `policy.tuf_enabled` field to `true` when installing Tanzu Application Platform. By default, keyless authorities support is deactivated.

By default, the public official Sigstore “The Update Framework (TUF) server” is used. You can use an alternative Sigstore Stack by setting `policy.tuf_mirror` and `policy.tuf_root`.

The Sigstore Stack consists of:

- Trillian
- Rekor
- Fulcio
- Certificate Transparency Log (CTLog)
- The Update Framework (TUF)

For an air-gapped environment, an internally accessible Sigstore Stack is required for keyless authorities.

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings. This is done by using the package manager installed by Tanzu Cluster Essentials.

For more information about profiles, see Components and installation profiles.

To prepare to install a profile:

1. List version information for the package by running:

```
tanzu package available list tap.tanzu.vmware.com --namespace tap-install
```

2. Create a `tap-values.yaml` file by using the Full Profile sample as a guide. These samples have the minimum configuration required to deploy Tanzu Application Platform. The sample values file contains the necessary defaults for:
    - The meta-package, or parent Tanzu Application Platform package
    - Subordinate packages, or individual child packages

    Keep the values file for future configuration use.

Full Profile

To install Tanzu Application Platform with Supply Chain Basic, you must retrieve your cluster's base64 encoded ca certificate from `$HOME/.kube/config`. Retrieve the `certificate-authority-data` from the respective cluster section and input it as `B64_ENCODED_CA` in the `tap-values.yaml`.

The following is the YAML file sample for the full-profile:

Important

Tanzu Build Service is installed by default with `lite` depndencies. When installing Tanzu Build Service in an air-gapped environment, the lite dependencies are not available because they require Internet access. You must install the `full` dependencies by setting `exclude_dependencies` to `true`.

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
    secret:
      name: "KP-DEFAULT-REPO-SECRET"
      namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----
profile: full
ceip_policy_disclosed: true
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section by default,
but can be overridden by setting a different value.
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  exclude_dependencies: true
supply_chain: basic
scanning:
  metadataStore:
    url: ""
contour:
  infrastructure_provider: aws
  envoy:
    service:
      type: LoadBalancer
      annotations:
      # This annotation is for air-gapped AWS only.
          service.kubernetes.io/aws-load-balancer-internal: "true"

ootb_supply_chain_basic:
  registry:
      server: "SERVER-NAME" # Takes the value from the shared section by default, but
can be overridden by setting a different value.
```

```
      repository: "REPO-NAME" # Takes the value from the shared section by default, bu
t can be overridden by setting a different value.
  gitops:
      ssh_secret: "SSH-SECRET"
  maven:
      repository:
         url: https://MAVEN-URL
         secret_name: "MAVEN-CREDENTIALS"

accelerator:
  ingress:
    include: true
    enable_tls: false
  git_credentials:
    secret_name: git-credentials
    username: GITLAB-USER
    password: GITLAB-PASSWORD

appliveview:
  ingressEnabled: true

appliveview_connector:
  backend:
    ingressEnabled: true
    sslDeactivated: false
    host: appliveview.INGRESS-DOMAIN
    caCertData: |-
      -----BEGIN CERTIFICATE-----
      MIIGMzCCBBugAwIBAgIJALHHzQjxM6wMMA0GCSqGSIb3DQEBDQUAMGcxCzAJBgNV
      BAgMAklOMRQwEgYDVQQHDAtNaW5uZWFwb2xpczEPMA0GA1UECgwGVk13YXJlMRMw
      -----END CERTIFICATE-----

tap_gui:
  app_config:
    kubernetes:
      serviceLocatorMethod:
        type: multiTenant
      clusterLocatorMethods:
        - type: config
          clusters:
            - url: https://${KUBERNETES_SERVICE_HOST}:${KUBERNETES_SERVICE_PORT}
              name: host
              authProvider: serviceAccount
              serviceAccountToken: ${KUBERNETES_SERVICE_ACCOUNT_TOKEN}
              skipTLSVerify: false
              caData: B64_ENCODED_CA
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
    #Example Integration for custom GitLab:
    integrations:
      gitlab:
        - host: GITLAB-URL
          token: GITLAB-TOKEN
          apiBaseUrl: https://GITLABURL/api/v4/
    backend:
      reading:
        allow:
          - host: GITLAB-URL # Example URL: gitlab.example.com

metadata_store:
  ns_for_export_app_cert: "MY-DEV-NAMESPACE"
  app_service_type: ClusterIP # Defaults to LoadBalancer. If shared.ingress_domain is
set earlier, this must be set to ClusterIP.
```

```
grype:
  namespace: "MY-DEV-NAMESPACE"
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-SECRET` is the secret with user credentials that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
  - You must create the secret before the installation. For example, you can use the `registry-credentials` secret created earlier.
- `KP-DEFAULT-REPO-SECRET-NAMESPACE` is the namespace where `KP-DEFAULT-REPO-SECRET` is created.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. If this key is passed through the shared section earlier and AWS ECR registry is used, you must ensure that the `SERVER-NAME/REPO-NAME/buildservice` and `SERVER-NAME/REPO-NAME/workloads` exist. AWS ECR expects the paths to be pre-created.
- Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET` is the secret name for https authentication, certificate authority, and SSH authentication. See Git authentication for more information.
- `MAVEN-CREDENTIALS` is the name of the secret with maven creds. This secret must be in the developer namespace. You can create it after the fact.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `GITLABURL` is the host name of your GitLab instance.
- `GITLAB-USER` is the user name of your GitLab instance.

- `GITLAB-PASSWORD` is the password for the `GITLAB-USER` of your GitLab instance. This can also be the `GITLAB-TOKEN`.
- `GITLAB-TOKEN` is the API token for your GitLab instance.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the SCST - Store CA certificate to all namespaces.

  Note: To install Grype in multiple namespaces, use a namespace provisioner. See Namespace Provisioner.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.

Note

The `appliveview_connector.backend.sslDisabled` key is deprecated and renamed to `appliveview_connector.backend.sslDeactivated`.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

TLS is enabled by default on Application Live View back end using ClusterIssuer. Set the `ingressEnabled` key to `true` for TLS to be enabled on Application Live View back end using ClusterIssuer. This key is set to `false` by default.

The `appliveview-cert` certificate is generated by default and its issuerRef points to the `.ingress_issuer` value. The `ingress_issuer` key consumes the value `shared.ingress_issuer` from `tap-values.yaml` by default when you don't specify the `ingress_issuer` in `tap-values.yaml`.

When `ingressEnabled` is `true`, an HTTPProxy object is created in the cluster and `appliveview-cert` certificate is generated by default in the `app_live_view` namespace. The secretName `appliveview-cert` stores this certificate.

To verify the HTTPProxy object with the secret, run:

```
kubectl get httpproxy -A
```

Expected output:

```
NAMESPACE            NAME
FQDN                                                        TLS SECRET
STATUS   STATUS DESCRIPTION
app-live-view        appliveview
appliveview.192.168.42.55.nip.io                            appliveview-cert   va
lid    Valid HTTPProxy
```

The `appliveview_connector.backend.host` key is the back end host in the view cluster. The `appliveview_connector.backend.caCertData` key is the certificate retrieved from the HTTPProxy secret exposed by Application Live View back end in the view cluster. To retrieve this certificate, run the following command in the view cluster:

```
kubectl get secret appliveview-cert -n app-live-view -o yaml |  yq '.data."ca.crt"' |
base64 -d
```

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v $TAP_VERSION --values-file
   tap-values.yaml -n tap-install
   ```

   Where `$TAP_VERSION` is the Tanzu Application Platform version environment variable you defined earlier.

2. Verify the package install by running:

   ```
   tanzu package installed get tap -n tap-install
   ```

   This may take 5-10 minutes because it installs several packages on your cluster.

3. Verify that all the necessary packages in the profile are installed by running:

   ```
   tanzu package installed list -A
   ```

Next steps

- Install the Tanzu Build Service dependencies

Install the Tanzu Build Service dependencies

This topic tells you how to install the Tanzu Build Service (TBS) full dependencies on Tanzu Application Platform (commonly known as TAP).

By default, Tanzu Build Service is installed with `lite` dependencies.

When installing Tanzu Build Service on an air-gapped environment, the `lite` dependencies cannot be used as they require Internet access. You must install the `full` dependencies.

To install `full` dependencies:

1. Get the latest version of the Tanzu Build Service package by running:

   ```
   tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
   all
   ```

2. Relocate the Tanzu Build Service `full` dependencies package repository by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
   ps-package-repo:VERSION \
     --to-tar=tbs-full-deps.tar
   # move tbs-full-deps.tar to environment with registry access
   imgpkg copy --tar tbs-full-deps.tar \
     --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps
   ```

   Where:

   - `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.
   - `INSTALL-REGISTRY-HOSTNAME` is your container registry.
   - `TARGET-REPOSITORY` is your target repository.

3. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add tbs-full-deps-repository \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps:VERSION \
  --namespace tap-install
```

Where:

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.
- `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

4. Install the `full` dependencies package by running:

```
tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
N -n tap-install
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

Next steps

- Configure custom CAs for Tanzu Application Platform GUI

Configure custom certificate authorities for Tanzu Application Platform GUI

This topic tells you how to configure your Tanzu Application Platform GUI (commonly known as TAP GUI) to trust unusual certificate authorities (CA) when making outbound connections.

Tanzu Application Platform GUI might require custom certificates when connecting to persistent databases or custom catalog locations that require SSL. You use overlays with PackageInstalls to make this possible. There are two ways to implement this workaround: you can add a custom CA or you can deactivate all SSL verification.

Add a custom CA

The overlay previously available in this section is no longer necessary. As of Tanzu Application Platform v1.3, the value `ca_cert_data` is supported at the top level of its values file. Any number of newline-delimited CA certificates in PEM format are accepted.

For example:

```
# tap-gui-values.yaml
ca_cert_data: |
  -----BEGIN CERTIFICATE-----
  cert data here
  -----END CERTIFICATE-----

  -----BEGIN CERTIFICATE-----
  other cert data here
  -----END CERTIFICATE-----
app_config:
  # ...
```

Tanzu Application Platform GUI also inherits `shared.ca_cert_data` from your `tap-values.yaml` file. `shared.ca_cert_data` is newline-concatenated with `ca_certs` given directly to Tanzu Application Platform GUI.

```
shared:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
```

```
    cert data here
    -----END CERTIFICATE-----

tap_gui:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    other cert data here
    -----END CERTIFICATE-----
  app_config:
    # ...
```

To verify that Tanzu Application Platform GUI has processed the custom CA certificates, check that the `ca-certs-data` volume with mount path `/etc/custom-ca-certs-data` is mounted in the Tanzu Application Platform GUI server pod.

Deactivate all SSL verification

To deactivate SSL verification to allow for self-signed certificates, set the Tanzu Application Platform GUI pod's environment variable as `NODE_TLS_REJECT_UNAUTHORIZED=0`. When the value equals `0`, certificate validation is deactivated for TLS connections.

To do this, use the `package_overlays` key in the Tanzu Application Platform values file. For instructions, see Customize Package Installation.

The following YAML is an example `Secret` containing an overlay to deactivate TLS:

```
apiVersion: v1
kind: Secret
metadata:
  name: deactivate-tls-overlay
  namespace: tap-install
stringData:
  deactivate-tls-overlay.yml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind":"Deployment", "metadata": {"name": "se
rver", "namespace": "NAMESPACE"}}),expects="1+"
    ---
    spec:
      template:
        spec:
          containers:
            #@overlay/match by=overlay.all,expects="1+"
            #@overlay/match-child-defaults missing_ok=True
            - env:
              - name: NODE_TLS_REJECT_UNAUTHORIZED
                value: "0"
```

Where `NAMESPACE` is the namespace in which your Tanzu Application Platform GUI instance is deployed. For example, `tap-gui`.

Next steps

- Configure Application Accelerator

Configure Application Accelerator

This topic describes advanced configuration options available for Application Accelerator. This includes configuring Git-Ops style deployments of accelerators and configurations for use with non-public repositories and in air-gapped environments.

Accelerators are created either using the Tanzu CLI or by applying a YAML manifest using kubectl. Another option is Using a Git-Ops style configuration for deploying a set of managed accelerators.

Application Accelerator pulls content from accelerator source repositories using either the “Flux SourceController” or the “Tanzu Application Platform Source Controller” components. If the repository used is accessible anonymously from a public server, you do not have to configure anything additional. Otherwise, provide authentication as explained in Using non-public repositories. There are also options for making these configurations easier explained in Configuring tap-values.yaml with Git credentials secret

Using a Git-Ops style configuration for deploying a set of managed accelerators

To enable a Git-Ops style of managing resources used for deploying accelerators, there is a new set of properties for the Application Accelerator configuration. The resources are managed using a Carvel kapp-controller App in the `accelerator-system` namespace that watches a Git repository containing the manifests for the accelerators. This means that you can make changes to the manifests, or to the accelerators they point to, and the changes are reconciled and reflected in the deployed resources.

You can specify the following accelerator configuration properties when installing the Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
accelerator:
  managed_resources:
    enable: true
    git:
      url: GIT-REPO-URL
      ref: origin/main
      sub_path: null
      secret_ref: git-credentials
```

Where:

- `GIT-REPO-URL` is the URL of a Git repository that contains manifest YAML files for the accelerators that you want to have managed. The URL must start with `https://` or `git@`. You can specify a `sub_path` if necessary and also a `secret_ref` if the repository requires authentication. If not needed, then leave these additional properties out.

  For more information, see Configure tap-values.yaml with Git credentials secret and Creating a manifest with multiple accelerators and fragments in this topic.

Functional and Organizational Considerations

Any accelerator manifest that is defined under the `GIT-REPO-URL` and optional `sub_path` is selected by the kapp-controller app. If there are multiple manifests at the defined `GIT-REPO-URL`, they are all watched for changes and displayed to the user as a merged catalog.

For example: if you have two manifests containing multiple accelerator or fragment definitions, `manifest-1.yaml`, and `manifest-2.yaml`, on the same path in the organizational considerations. The resulting catalog is (`manifest-1.yaml` + `manifest-2.yaml`).

Examples for creating accelerators

A minimal example for creating an accelerator

A minimal example might look like the following manifest:

`spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: main
```

This example creates an accelerator named `spring-cloud-serverless`. The `displayName`, `description`, `iconUrl`, and `tags` text boxes are populated based on the content under the `accelerator` key in the `accelerator.yaml` file found in the `main` branch of the Git repository at Application Accelerator Samples under the sub-path `spring-cloud-serverless`. For example:

`accelerator.yaml`

```
accelerator:
  displayName: Spring Cloud Serverless
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.p
ng
  tags:
  - java
  - spring
  - cloud
  - function
  - serverless
  - tanzu
...
```

To create this accelerator with kubectl, run:

```
kubectl apply --namespace --accelerator-system --filename spring-cloud-serverless.yaml
```

Or, you can use the Tanzu CLI and run:

```
tanzu accelerator create spring-cloud-serverless --git-repo https://github.com/vmware-
tanzu/application-accelerator-samples.git --git-branch main --git-sub-path spring-clou
d-serverless
```

An example for creating an accelerator with customized properties

You can specify the `displayName`, `description`, `iconUrl`, and `tags` text boxes and this overrides any values provided in the accelerator's Git repository. The following example explicitly sets those text boxes and the `ignore` text box:

`my-spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: my-spring-cloud-serverless
spec:
  displayName: My Spring Cloud Serverless
  description: My own Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.p
ng
  tags:
    - spring
    - cloud
```

```
    - function
    - serverless
  git:
    ignore: ".git/, bin/"
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: test
```

To create this accelerator with kubectl, run:

```
kubectl apply --namespace --accelerator-system --filename my-spring-cloud-serverless.y
aml
```

To use the Tanzu CLI, run:

```
tanzu accelerator create my-spring-cloud-serverless --git-repo https://github.com/vmwa
re-tanzu/application-accelerator-samples --git-branch main --git-sub-path spring-cloud
-serverless \
  --description "My own Spring Cloud Function serverless app" \
  --display-name "My Spring Cloud Serverless" \
  --icon-url https://raw.githubusercontent.com/simple-starters/icons/master/icon-clou
d.png \
  --tags "spring,cloud,function,serverless"
```

Note

It is not possible to provide the `git.ignore` option with the Tanzu CLI.

Creating a manifest with multiple accelerators and fragments

You might have a manifest that contains multiple accelerators or fragments. For example:

`accelerator-collection.yaml`

```
---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: main
---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: tanzu-java-web-app
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: tanzu-java-web-app
    ref:
      branch: main
```

For a larger example of this, see Sample Accelerators Main. Optionally, use this to create an initial catalog of accelerators and fragments during a fresh Application Accelerator install.

Configure `tap-values.yaml` with Git credentials secret

Note

For how to create a new OAuth Token for optional Git repository creation, see Create an Application Accelerator Git repository during project creation.

When deploying accelerators using Git repositories that requires authentication or are installed with custom CA certificates, you must provide some additional authentication values in a secret. The examples in the next section provide more details. This section describes how to configure a Git credentials secret that is used in later Git-based examples.

You can specify the following accelerator configuration properties when installing Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
accelerator:
  git_credentials:
    secret_name: git-credentials
    username: GIT-USER-NAME
    password: GIT-CREDENTIALS
    ca_file: CUSTOM-CA-CERT
```

Where:

- `GIT-USER-NAME` is the user name for authenticating with the Git repository.
- `GIT-CREDENTIALS` is the password or access token used for authenticating with the Git repository. VMware recommends using an access token for this.
- `CUSTOM-CA-CERT` is the certificate data needed when accessing the Git repository.

This is an example of this part of a `tap-values.yaml` configuration:

```
accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret
    ca_file: |
      -----BEGIN CERTIFICATE-----
      .
      .
      .  < certificate data >
      .
      .
      -----END CERTIFICATE-----
```

You can specify the custom CA certificate data using the shared config value `shared.ca_cert_data` and it propagates to all components that can make use of it, including the App Accelerator configuration. The example earlier produces an output such as this using the shared value:

```
shared:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    .
    .
    .  < certificate data >
    .
    .
    -----END CERTIFICATE-----
```

```
accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret
```

Using non-public repositories

For GitHub repositories that aren't accessible anonymously, you must provide credentials in a Secret.

- For HTTPS repositories the secret must contain user name and password fields. The password field can contain a personal access token instead of an actual password. For more information, see Fluxcd/source-controller basic access authentication.
- For HTTPS with self-signed certificates, you can add a `.data.caFile` value to the secret created for HTTPS authentication. For more information, see fluxcd/source-controller HTTPS Certificate Authority.
- For SSH repositories, the secret must contain identity, identity.pub, and known_hosts text boxes. For more information, see fluxcd/source-controller SSH authentication.
- For Image repositories that aren't publicly available, an image pull secret might be provided. For more information, see Kubernetes documentation on using imagePullSecrets.

Examples for a private Git repository

Example using http credentials

To create an accelerator using a private Git repository, first create a secret with the HTTP credentials.

Note

For better security, use an access token as the password.

```
kubectl create secret generic https-credentials \
    --namespace accelerator-system \
    --from-literal=username=<user> \
    --from-literal=password=<access-token>
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you created and verified the secret, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-credentials
```

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Example using http credentials with self-signed certificate

To create an accelerator using a private Git repository with a self-signed certificate, create a secret with the HTTP credentials and the certificate.

Note

For better security, use an access token as the password.

```
kubectl create secret generic https-ca-credentials \
    --namespace accelerator-system \
    --from-literal=username=<user> \
    --from-literal=password=<access-token> \
    --from-file=caFile=<path-to-CA-file>
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-ca-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-ca-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
  caFile: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-ca-credentials
```

Important

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Example using SSH credentials

To create an accelerator using a private Git repository, create a secret with the SSH credentials such as this example:

```
ssh-keygen -q -N "" -f ./identity
ssh-keyscan github.com > ./known_hosts
kubectl create secret generic ssh-credentials \
    --namespace accelerator-system \
    --from-file=./identity \
    --from-file=./identity.pub \
    --from-file=./known_hosts
```

If you have a key file already created, skip the `ssh-keygen` and `ssh-keyscan` steps and replace the values for the `kubectl create secret` command. Such as:

- `--from-file=identity=<path to your identity file>`
- `--from-file=identity.pub=<path to your identity.pub file>`
- `--from-file=known_hosts=<path to your know_hosts file>`

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system ssh-credentials -o yaml
```

The output is similar to :

```
apiVersion: v1
kind: Secret
metadata:
  name: ssh-credentials
  namespace: accelerator-system
type: Opaque
data:
  identity: <BASE64>
  identity.pub: <BASE64>
  known_hosts: <BASE64>
```

To use this secret when creating an accelerator, provide the secret name in the `spec.git.secretRef.name` property:

`private-acc-ssh.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: ssh-credentials
```

When using SSH credentials, the `REPOSITORY-URL` must include the user name as part of the URL. For example: `ssh://user@example.com:22/repository.git`. For more information, see Flux documentation.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Examples for a private source-image repository

If your registry uses a self-signed certificate then you must add the CA certificate data to the configuration for the “Tanzu Application Platform Source Controller” component. Add it under `source_controller.ca_cert_data` in your `tap-values.yaml` file that is used during installation.

`tap-values.yaml`

```
source_controller:
  ca_cert_data: |-
    -----BEGIN CERTIFICATE-----
    .
    .
    .  < certificate data >
    .
    .
    -----END CERTIFICATE-----
```

Example using image-pull credentials

To create an accelerator using a private source-image repository, create a secret with the image-pull credentials:

```
create secret generic registry-credentials \
    --namespace accelerator-system \
    --from-literal=username=<user> \
    --from-literal=password=<password>
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system registry-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: registry-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  source:
    image: "registry.example.com/test/private-acc-src:latest"
    imagePullSecrets:
    - name: registry-credentials
```

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Configure ingress timeouts when some accelerators take longer to generate

If Tanzu Application Platform is configured to use an ingress for Tanzu Application Platform GUI and the Accelerator Server, then it might detect a timeout during accelerator generation. This can happen if the accelerator takes a longer time to generate than the default timeout. When this happens, Tanzu Application Platform GUI appears to continue to run for an indefinite period. In the IDE extension, it shows a `504` error. To mitigate this, you can increase the timeout value for the HTTPProxy resources used for the ingress by applying secrets with overlays to edit the HTTPProxy resources.

Configure an ingress timeout overlay secret for each HTTPProxy

For Tanzu Application Platform GUI, create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-tap-gui-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "tap-
gui"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "server"}]})
```

```
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

For Accelerator Server (used for IDE extension), create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-accelerator-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "acce
lerator"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "acc-server"}]})
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

Apply the timeout overlay secrets in tap-values.yaml

Add the following `package_overlays` section to `tap-values.yaml` before installing or updating Tanzu Application Platform:

```
package_overlays:
- name: tap-gui
  secrets:
  - name: patch-tap-gui-timeout
- name: accelerator
  secrets:
  - name: patch-accelerator-timeout
```

Configuring skipping TLS verification for access to Source Controller

You can configure the Flux or Tanzu Application Platform Source Controller to use Transport Layer Security (TLS) and use custom certificates. In that case, configure the Accelerator System to skip the TLS verification for calls to access the sources by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```
sources:
  skip_tls_verify: true
```

Enabling TLS for Accelerator Server

To enable TLS for the Accelerator Server, the following properties must be provided in the `accelerator` section of the `tap-values.yaml` file:

```
server:
  tls:
    enabled: true
```

```
    key: SERVER-PRIVATE-KEY
    crt: SERVER-CERTIFICATE
```

Where:

- `SERVER-PRIVATE-KEY` is the pem encoded server private key.
- `SERVER-CERTIFICATE` is the pem encoded server certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Server:

```
server:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
      .  < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      .  < certificate data >
      .
      -----END CERTIFICATE-----
```

Configuring skipping TLS verification of Engine calls for Accelerator Server

If you configure the Accelerator Engine to use TLS and use custom certificates, then you can configure the Accelerator Server to skip the TLS verification for calls to the Engine by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```
server:
  engine_skip_tls_verify: true
```

Enabling TLS for Accelerator Engine

To enable TLS for the Accelerator Engine, the following properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
engine:
  tls:
    enabled: true
    key: ENGINE-PRIVATE-KEY
    crt: ENGINE-CERTIFICATE
```

Where:

- `ENGINE-PRIVATE-KEY` is the pem encoded acc-engine private key.
- `ENGINE-CERTIFICATE` is the pem encoded acc-engine certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Engine:

```
engine:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
```

```
      .  < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      .  < certificate data >
      .
      -----END CERTIFICATE-----
```

Next steps

- Using Grype in offline and air-gapped environments

Use Grype in offline and air-gapped environments

The `grype` CLI attempts to perform two over the Internet calls:

- One to verify for later versions of the CLI.
- One to update the vulnerability database before scanning.

For the `grype` CLI to function in an offline or air-gapped environment, the vulnerability database must be hosted within the environment. You must configure the `grype` CLI with the internal URL.

The `grype` CLI accepts environment variables to satisfy these needs.

Host the Grype vulnerability database

To host Grype's vulnerability database in an air-gapped environment:

1. Retrieve Grype's listing file from its public endpoint: https://toolbox-data.anchore.io/grype/databases/listing.json.
2. Create your own `listing.json` file.

   Note Different Grype versions require specific database schema versions. To avoid compatibility issues between different versions, include a database schema for each version. For example:

   ```
       {
         "available": {
           "1": [
             {
               "built": "2023-06-16T01:33:30Z",
               "version": 1,
               "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
   ty-db_v1_2023-06-16T01:33:30Z_1621f4169ffd15bea9e5.tar.gz",
               "checksum": "sha256:3f2c1b432945cca9a69b2e604f6fb231fec450fdd27f494
   6fc5608692b63a9d1"
             }
           ],
           "2": [
             {
               "built": "2023-06-16T01:33:30Z",
               "version": 2,
               "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
   ty-db_v2_2023-06-16T01:33:30Z_d6eee5e78d9b78285e1a.tar.gz",
               "checksum": "sha256:7b7e3a2a7712c72b8c5cc777733c4d8d140d8cfee65e4f0
   4540abbdfe3ef1f65"
             }
           ],
           "3": [
   ```

```
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 3,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v3_2023-06-16T01:33:30Z_f96ae38a7b05987c3ece.tar.gz",
            "checksum": "sha256:8ea9fae3fda3bf3bf35bd5e5eb656fc127b59cd3c42db4c
36795556aab8a9cf0"
          }
        ],
        "4": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 4,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v4_2023-06-16T01:33:30Z_13bba2fa8ff62b7f8b26.tar.gz",
            "checksum": "sha256:3b53d20241b88e5aa45feb817b325c53d6efbe9fa1fc5a6
7eeddaecafa7687e0"
          }
        ],
        "5": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 5,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v5_2023-06-16T01:33:30Z_e07da3853f6db6eb1104.tar.gz",
            "checksum": "sha256:93d4d9d2f9e39f86570f832cf85b7149a949ca6f1613581
b10c12393509d884f"
          }
        ]
      }
    }
```


Where `url` points to a tarball containing Grype's `vulnerability`, `db`, and `metadata.json` files.

3. Download and host the tarballs in your internal file server.

Note

Some storage solutions for internal file servers change the name of TAR files automatically because of their limits. Notice these modified names and reflect the changes in the `url`. Ensure that the timestamp in the name is correctly formatted because Grype parses the name of TAR artifact to get the timestamp.

4. Update the download `url` to point at your internal endpoint.

   For information about setting up an offline vulnerability database, see the Anchore Grype README in GitHub.

To enable Grype in offline air-gapped environments

1. Add the following to your `tap-values.yaml` file:

```
grype:
  db:
    dbUpdateUrl: INTERNAL-VULN-DB-URL
```

   Where `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

2. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Configure Grype environmental variables

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-environmental-variables
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

Where `spec.template.initContainers[]` specifies setting one or more environment variables in the `scan-plugin` initContainer.

Note

If you are using the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, you must import the overlay `Secret`. See Import overlay secrets.

Troubleshooting

ERROR failed to fetch latest cli version

Note

This message is a warning and the Grype scan still runs with this message.

The Grype CLI checks for later versions of the CLI by contacting the anchore endpoint over the Internet.

```
ERROR failed to fetch latest version: Get "https://toolbox-data.anchore.io/grype/releases/latest/VERSION": dial tcp: lookup toolbox-data.anchore.io on [::1]:53: read udp [::1]:65010->[::1]:53: read: connection refused
```

Solution

To deactivate this check, set the environment variable `GRYPE_CHECK_FOR_APP_UPDATE` to `false` by using a package overlay with the following steps:

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-deactivate-cli-check-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```
package_overlays:
  - name: "grype"
    secrets:
        - name: "grype-airgap-deactivate-cli-check-overlay"
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Database is too old

```
1 error occurred:
  * db could not be loaded: the vulnerability database was built N days/weeks ago (max
allowed age is 5 days)
```

Grype needs up-to-date vulnerability information to provide accurate matches. By default, it fails to run if the local database was not built in the last 5 days.

Solution

Two options to resolve this:

1. Stale databases weaken your security posture. VMware recommends updating the database daily as the first recommended solution.
2. If updating the database daily is not an option, the data staleness check is configurable by using the environment variable `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` and is addressed using a package overlay with the following steps:

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-override-stale-db-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1
+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), exp
ects="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_DB_MAX_ALLOWED_BUILT_AGE #! see note on best
practices
                value: "120h"
```

Note

The default maximum allowed built age of Grype's vulnerability database is 5 days. This means that scanning with a 6 day old database causes the scan to fail. You can use the `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` parameter to override the default in accordance with your security posture.

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```
package_overlays:
  - name: "grype"
    secrets:
        - name: "grype-airgap-override-stale-db-overlay"
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Vulnerability database is invalid

```
scan-pod[scan-plugin]  1 error occurred:
scan-pod[scan-plugin]  * failed to load vulnerability db: vulnerability database is in
valid (run db update to correct): database metadata not found: /.cache/grype/db/5
```

Solution

Examine the `listing.json` file you created. This matches the format of the listing file. The listing file is located at Anchore Grype's public endpoint. See the Grype README.md in GitHub.

An example `listing.json`:

```
{
  "available": {
    "5": [
      {
        "built": "2023-03-28T01:29:38Z",
        "version": 5,
        "url": "https://toolbox-data.anchore.io/grype/databases/vulnerability-db_v5_20
23-03-28T01:29:38Z_e49d318c32a6113eed07.tar.gz",
        "checksum": "sha256:408ce2932f04dee929a5df524e92494f2d635c6b19e30ff9f0a50425b1
fc29a1"
      },
      .....
    ]
  }
}
```

Where:

- `5` refers to the Grype's vulnerability database schema.
- `built` is the build timestamp in the format `yyyy-MM-ddTHH:mm:ssZ`.
- `url` is the download URL for the tarball containing the database. This points at your internal endpoint. The tarball contains the following files:
    - `vulnerability.db` is an SQLite file that is Grype's vulnerability database. Each time the data shape of the vulnerability database changes, a new schema is created. Different Grype versions require specific database schema versions. For example, Grype `v0.54.0` requires database schema version v5.
    - `metadata.json` file
- `checksum` is the SHA used to verify the database's integrity.

Verify these possible reasons why the vulnerability database is not valid:

1. The database schema is invalid. Confirm that the required database schema for the installed Grype version is used. Confirm that the top level version key matches the nested `version`. For example, the top level version `1` in the following snippet does not match the nested `version: 5`.

    ```
    {
      "available": {
        "1": [{
                "built": "2023-02-08T08_17_20Z",
                "version": 5,
                "url": "https://INTERNAL-ENDPOINT/PATH-TO-TARBALL/vulnerability-db_v
    5_2023-02-08T08_17_20Z_6ef73016d160043c630f.tar.gz",
                "checksum": "sha256:aab8d369933c845878ef1b53bb5c26ee49b91ddc5cd87c9e
    b57ffb203a88a72f"
        }]
      }
    }
    ```

    Where `PATH-TO-TARBALL` is the path to the tarball containing the vulnerability database.

    As stale databases weaken your security posture, VMware recommends using the newest entry of the relevant schema version in the `listing.json` file. See Anchore's grype-db in GitHub.

2. The `built` parameters in the `listing.json` file are incorrectly formatted. The proper format is `yyyy-MM-ddTHH:mm:ssZ`.

3. The `url` that you modified to point at an internal endpoint is not reachable from within the cluster. For information about verifying connectivity, see Debug Grype database in a cluster.
4. Verify if there are syntax errors in the listing.json:

```
grype db check
```

5. Validate the configured listing.json:

```
grype db list -o raw
```

Debug Grype database in a cluster

1. Describe the failed source scan or image scan to verify the name of the ScanTemplate being used.
    - For `sourcescan`, run:

      ```
      kubectl describe sourcescan SCAN-NAME -n DEV-NAMESPACE
      ```

    - For `imagescan`, run:

      ```
      kubectl describe imagescan SCAN-NAME -n DEV-NAMESPACE
      ```

    Where `SCAN-NAME` is the name of the source or image scan that failed.
2. Pause reconciliation of the `grype.scanning.apps.tanzu.vmware.com` package:

```
kctrl package installed pause -i <PACKAGE-INSTALL-NAME> -n tap-install
```

    Where `PACKAGE-INSTALL-NAME` is the name of the `grype.scanning.apps.tanzu.vmware.com` package (e.g. grype)
3. Edit the ScanTemplate's `scan-plugin` container to include a "sleep" entrypoint which allows you to troubleshoot inside the container:

```
- name: scan-plugin
  volumeMounts:
    ...
  image: #@ data.values.scanner.image
  imagePullPolicy: IfNotPresent
  env:
    ...
  command: ["/bin/bash"]
  args:
  - "sleep 1800" # insert 30 min sleep here
```

4. Re-run the scan.
5. Get the name of the `scan-plugin` pod.

```
kubectl get pods -n DEV-NAMESPACE
```

6. Get a shell to the container.

```
kubectl exec --stdin --tty SCAN-PLUGIN-POD -c step-scan-plugin -- /bin/bash
```

    Where `SCAN-PLUGIN-POD` is the name of the `scan-plugin` pod. For more information, see the Kubernetes documentation.

7. Inside the container, run Grype CLI commands to report database status and verify connectivity from the cluster to the mirror. See the Grype documentation in GitHub.
    - Report current status of Grype's database, such as location, build date, and checksum:

      ```
      grype db status
      ```

8. Ensure that the built parameters in the `listing.json` has timestamps in this proper format `yyyy-MM-ddTHH:mm:ssZ`.
9. After you complete troubleshooting, use the following command to trigger reconciliation:

   ```
   kctrl package installed kick -i <PACKAGE-INSTALL-NAME> -n tap-install
   ```

   Where `PACKAGE-INSTALL-NAME` is the name of the `grype.scanning.apps.tanzu.vmware.com` package, such as Grype.

Grype package overlays are not applied to scantemplates created by Namespace Provisioner

If you used the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, see Import overlay secrets.

Set up developer namespaces to use your installed packages

For details about how to automatically set up your developer namespaces, see Provision developer namespaces in Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see Legacy namespace setup.

Next steps

For more information about the Namespace Provisioner mode, see Work with Git repositories in air-gapped environments with Namespace Provisioner.

For more information about the manual mode, see Deploy an air-gapped workload.

Install Tanzu Application Platform (AWS)

You can install Tanzu Application Platform (commonly known as TAP) on Amazon Elastic Kubernetes Services (EKS) by using Amazon Elastic Container Registry (ECR).

To install, take the following steps.

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create AWS Resources (EKS Cluster, roles, etc)	Create AWS Resources
4.	Install Cluster Essentials for Tanzu	Deploy Cluster Essentials
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	Install the Tanzu Application Platform package and profiles
6.	(Optional) Install any additional packages that were not in the profile	Install individual packages
7.	Set up developer namespaces to use your installed packages	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE)	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on your Kubernetes clusters, get started with Tanzu Application Platform and create your ECR repositories for your workload, such as `tanzu-application-platform/tanzu-java-web-app-default`, `tanzu-application-platform/tanzu-java-web-app-default-bundle`, and `tanzu-application-platform/tanzu-java-web-app-default-source`.

Install Tanzu Application Platform (AWS)

You can install Tanzu Application Platform (commonly known as TAP) on Amazon Elastic Kubernetes Services (EKS) by using Amazon Elastic Container Registry (ECR).

To install, take the following steps.

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create AWS Resources (EKS Cluster, roles, etc)	Create AWS Resources
4.	Install Cluster Essentials for Tanzu	Deploy Cluster Essentials
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	Install the Tanzu Application Platform package and profiles
6.	(Optional) Install any additional packages that were not in the profile	Install individual packages
7.	Set up developer namespaces to use your installed packages	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE)	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on your Kubernetes clusters, get started with Tanzu Application Platform and create your ECR repositories for your workload, such as `tanzu-application-platform/tanzu-java-web-app-default`, `tanzu-application-platform/tanzu-java-web-app-default-bundle`, and `tanzu-application-platform/tanzu-java-web-app-default-source`.

Create AWS Resources for Tanzu Application Platform

To install Tanzu Application Platform (commonly known as TAP) within the Amazon Web Services (AWS) Ecosystem, you must create several AWS resources. Use this topic to learn how to create:

- An Amazon Elastic Kubernetes Service (EKS) cluster to install Tanzu Application Platform.
- Identity and Access Management (IAM) roles to allow authentication and authorization to read and write from Amazon Elastic Container Registry (ECR).
- ECR Repositories for the Tanzu Application Platform container images.

Creating these resources enables Tanzu Application Platform to use an IAM role bound to a Kubernetes service account for authentication, rather than the typical username and password stored in a Kubernetes secret strategy. For more information, see this AWS documentation.

This is important when using ECR because authenticating to ECR is a two-step process:

1. Retrieve a token using your AWS credentials.
2. Use the token to authenticate to the registry.

To increase security, the token has a lifetime of 12 hours. This makes storing it as a secret for a service impractical because it has to be refereshed every 12 hours.

Using an IAM role on a service account mitigates the need to retrieve the token at all because it is handled by credential helpers within the services.

Prerequisites

Before installing Tanzu Application Platform on AWS, you need:

- An AWS Account. You need to create all of your resources within Amazon Web Services, so you need an Amazon account. For more information, see How do I create and activate a new AWS account?. You need your account ID for this walkthrough.
- AWS CLI. This walkthrough uses the AWS CLI to both query and configure resources in AWS, such as IAM roles. For more information, see this AWS documentation.
- `eksctl` command line. The `eksctl` command line helps you manage the life cycle of EKS clusters. This guide uses it to create clusters. To install `eksctl`, see the eksctl documentation.

Export environment variables

Variables are used throughout this guide. To simplify the process and minimize the opportunity for errors, export these variables:

```
export AWS_ACCOUNT_ID=012345678901
export AWS_REGION=us-west-2
export EKS_CLUSTER_NAME=tap-on-aws
```

Where:

Variable	Description
AWS_ACCOUNT_ID	Your AWS account ID
AWS_REGION	The AWS region you are going to deploy to
EKS_CLUSTER_NAME	The name of your EKS Cluster

Create an EKS cluster

To create an EKS cluster in the specified region, run:

```
eksctl create cluster --name $EKS_CLUSTER_NAME --managed --region $AWS_REGION --instan
ce-types t3.xlarge --version 1.24 --with-oidc -N 5
```

Creating the control plane and node group can take anywhere from 30-60 minutes.

Note

This step is optional if you already have an existing EKS Cluster v1.23 or later with OpenID Connect (OIDC) authentication enabled. For more information about how to enable the OIDC provider, see AWS documentation.

Install EBS CSI driver

As a requirement for Tanzu Application Platform, EBS CSI driver is no longer installed by default starting from EKS 1.23. For more information about how to install EBS CSI driver, see AWS documentation.

Create the container repositories

ECR requires that the container repositories are already created. For Tanzu Application Platform, you need to create two repositories:

- A repository to store the Tanzu Application Platform service container images.
- A repository to store Tanzu Build Service Base OS and Buildpack container images.

To create these repositories, run:

```
aws ecr create-repository --repository-name tap-images --region $AWS_REGION
aws ecr create-repository --repository-name tap-build-service --region $AWS_REGION
```

Name the repositories any name you want, but remember the names for when you later build the configuration.

Create the workload container repositories

Similar to the two repositories created earlier for the platform, you must create repositories for each workload that Tanzu Application Platform creates before creating any workloads so that a repository is available to upload container images and workload bundles. This is because AWS ECR does not support automatically creating container repositories on initial push. For more information, see the AWS repository in GitHub.

When installing Tanzu Application Platform, you must specify a prefix for all workload registries. This topic uses `tanzu-application-platform` as the default value, but you can customize this value in the profile configuration created in Install Tanzu Application Platform package and profiles on AWS.

To use the default value, create two workload repositories for each workload with the following format:

```
tanzu-application-platform/WORKLOADNAME-NAMESPACE
tanzu-application-platform/WORKLOADNAME-NAMESPACE-bundle
```

For example, to create these repositories for the sample workload `tanzu-java-web-app` in the `default` namespace, you can run the following ECR command:

```
aws ecr create-repository --repository-name tanzu-application-platform/tanzu-java-web-
app-default --region $AWS_REGION
```

```
aws ecr create-repository --repository-name tanzu-application-platform/tanzu-java-web-
app-default-bundle --region $AWS_REGION
```

Note

The default Supply Chain Choreographer method of storing Kubernetes configuration is RegistryOps, which requires the `bundle` repository. If you enabled the GitOps capability, this repository is not required. For more information about the differences between RegistryOps and GitOps, see Use GitOps or RegistryOps with Supply Chain Choreographer.

Create IAM roles

By default, the EKS cluster is provisioned with an EC2 instance profile that provides read-only access for the entire EKS cluster to the ECR registry within your AWS account. For more information, see this AWS documentation.

However, some of the services within Tanzu Application Platform require write access to the container repositories. To provide that access, create IAM roles and add the ARN to the Kubernetes service accounts that those services use. This ensures that only the required services have access to write container images to ECR, rather than a blanket policy that applies to the entire cluster.

You must create two IAM Roles:

- Tanzu Build Service: Gives write access to the repository to allow the service to automatically upload new images. This is limited in scope to the service account for kpack and the dependency updater.
- Workload: Gives write access to the entire ECR registry with a prepended path. This prevents you from having to update the policy for each new workload created.

To create the roles, you must establish two policies:

- Trust Policy: Limits the scope to the OIDC endpoint for the Kubernetes cluster and the Kubernetes service account you attach the role to.
- Permission Policy: Limits the scope of actions the role can take on resources.

Note

These policies attempt to achieve a least privilege model. Review them to confirm they adhere to your organization's policies.

To simplify this walkthrough, use a script to create these policy documents and the roles. This script outputs the files and then creates the IAM roles by using the policy documents.

Run:

```
# Retrieve the OIDC endpoint from the Kubernetes cluster and store it for use in the p
olicy.
export OIDCPROVIDER=$(aws eks describe-cluster --name $EKS_CLUSTER_NAME --region $AWS_
REGION --output json | jq '.cluster.identity.oidc.issuer' | tr -d '"' | sed 's/http
s:\/\///')
cat << EOF > build-service-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDCPROVI
DER}"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "${OIDCPROVIDER}:aud": "sts.amazonaws.com"
                },
                "StringLike": {
                    "${OIDCPROVIDER}:sub": [
                        "system:serviceaccount:kpack:controller",
                        "system:serviceaccount:build-service:dependency-updater-contro
ller-serviceaccount"
                    ]
                }
            }
        }
    ]
}
EOF

cat << EOF > build-service-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecr:DescribeRegistry",
                "ecr:GetAuthorizationToken",
                "ecr:GetRegistryPolicy",
                "ecr:PutRegistryPolicy",
                "ecr:PutReplicationConfiguration",
                "ecr:DeleteRegistryPolicy"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TAPEcrBuildServiceGlobal"
        },
        {
            "Action": [
                "ecr:DescribeImages",
                "ecr:ListImages",
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:BatchGetRepositoryScanningConfiguration",
                "ecr:DescribeImageReplicationStatus",
                "ecr:DescribeImageScanFindings",
                "ecr:DescribeRepositories",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:GetRegistryScanningConfiguration",
                "ecr:GetRepositoryPolicy",
                "ecr:ListTagsForResource",
                "ecr:TagResource",
                "ecr:UntagResource",
                "ecr:BatchDeleteImage",
                "ecr:BatchImportUpstreamImage",
                "ecr:CompleteLayerUpload",
                "ecr:CreatePullThroughCacheRule",
                "ecr:CreateRepository",
                "ecr:DeleteLifecyclePolicy",
                "ecr:DeletePullThroughCacheRule",
```

```
                "ecr:DeleteRepository",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:PutImageScanningConfiguration",
                "ecr:PutImageTagMutability",
                "ecr:PutLifecyclePolicy",
                "ecr:PutRegistryScanningConfiguration",
                "ecr:ReplicateImage",
                "ecr:StartImageScan",
                "ecr:StartLifecyclePolicyPreview",
                "ecr:UploadLayerPart",
                "ecr:DeleteRepositoryPolicy",
                "ecr:SetRepositoryPolicy"
            ],
            "Resource": [
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-build-serv
ice",
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-images"
            ],
            "Effect": "Allow",
            "Sid": "TAPEcrBuildServiceScoped"
        }
    ]
}
EOF

cat << EOF > workload-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecr:DescribeRegistry",
                "ecr:GetAuthorizationToken",
                "ecr:GetRegistryPolicy",
                "ecr:PutRegistryPolicy",
                "ecr:PutReplicationConfiguration",
                "ecr:DeleteRegistryPolicy"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "TAPEcrWorkloadGlobal"
        },
        {
            "Action": [
                "ecr:DescribeImages",
                "ecr:ListImages",
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:BatchGetRepositoryScanningConfiguration",
                "ecr:DescribeImageReplicationStatus",
                "ecr:DescribeImageScanFindings",
                "ecr:DescribeRepositories",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:GetRegistryScanningConfiguration",
                "ecr:GetRepositoryPolicy",
                "ecr:ListTagsForResource",
                "ecr:TagResource",
                "ecr:UntagResource",
                "ecr:BatchDeleteImage",
                "ecr:BatchImportUpstreamImage",
                "ecr:CompleteLayerUpload",
                "ecr:CreatePullThroughCacheRule",
                "ecr:CreateRepository",
```

```
                "ecr:DeleteLifecyclePolicy",
                "ecr:DeletePullThroughCacheRule",
                "ecr:DeleteRepository",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:PutImageScanningConfiguration",
                "ecr:PutImageTagMutability",
                "ecr:PutLifecyclePolicy",
                "ecr:PutRegistryScanningConfiguration",
                "ecr:ReplicateImage",
                "ecr:StartImageScan",
                "ecr:StartLifecyclePolicyPreview",
                "ecr:UploadLayerPart",
                "ecr:DeleteRepositoryPolicy",
                "ecr:SetRepositoryPolicy"
            ],
            "Resource": [
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tap-build-serv
ice",
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tanzu-applicat
ion-platform/tanzu-java-web-app",
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tanzu-applicat
ion-platform/tanzu-java-web-app-bundle",
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tanzu-applicat
ion-platform",
                "arn:aws:ecr:${AWS_REGION}:${AWS_ACCOUNT_ID}:repository/tanzu-applicat
ion-platform/*"
            ],
            "Effect": "Allow",
            "Sid": "TAPEcrWorkloadScoped"
        }
    ]
}
EOF

cat << EOF > workload-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::${AWS_ACCOUNT_ID}:oidc-provider/${OIDCPROVI
DER}"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "${OIDCPROVIDER}:sub": "system:serviceaccount:*:default",
                    "${OIDCPROVIDER}:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
EOF


# Create the Tanzu Build Service Role
aws iam create-role --role-name tap-build-service --assume-role-policy-document fil
e://build-service-trust-policy.json
# Attach the Policy to the Build Role
aws iam put-role-policy --role-name tap-build-service --policy-name tapBuildServicePol
icy --policy-document file://build-service-policy.json

# Create the Workload Role
```

```
aws iam create-role --role-name tap-workload --assume-role-policy-document file://work
load-trust-policy.json
# Attach the Policy to the Workload Role
aws iam put-role-policy --role-name tap-workload --policy-name tapWorkload --policy-do
cument file://workload-policy.json
```

Install Tanzu Application Platform package and profiles on AWS

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) packages from your Tanzu Application Platform package repository on to AWS.

Before installing the packages, ensure you have:

- Completed the Prerequisites.
- Created AWS Resources.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.
- Installed Cluster Essentials for Tanzu.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform will depend on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

This section describes how to relocate images to the `tap-images` repository created in Amazon ECR. See Creating AWS Resources for more information.

To relocate images from the VMware Tanzu Network registry to the ECR registry:

1. Set up environment variables for installation use by running:

   ```
   export AWS_ACCOUNT_ID=MY-AWS-ACCOUNT-ID
   export AWS_REGION=TARGET-AWS-REGION
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export IMGPKG_REGISTRY_HOSTNAME_1=$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaw
   s.com
   export IMGPKG_REGISTRY_USERNAME_1=AWS
   export IMGPKG_REGISTRY_PASSWORD_1=`aws ecr get-login-password --region $AWS_REG
   ION`
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REGISTRY_HOSTNAME=$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.
   com
   export INSTALL_REPO=tap-images
   ```

   Where:

   - `MY-AWS-ACCOUNT-ID` is the account ID you deploy Tanzu Application Platform in. No dashes and must be in the format `012345678901`.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.

- `TARGET-AWS-REGION` is the region you deploy the Tanzu Application Platform to.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`

2. Install the Carvel tool imgpkg CLI.
3. Relocate the images with the `imgpkg` CLI by running:

```
imgpkg copy --concurrency 1 -b registry.tanzu.vmware.com/tanzu-application-plat
form/tap-packages:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTA
LL_REPO}
```

4. Create a namespace called `tap-install` for deploying any component packages by running:

```
kubectl create ns tap-install
```

This namespace keeps the objects grouped together logically.

5. (Optional) If you haven't relocated the images to ECR, create a secret to your registry by running:

```
tanzu secret registry add tap-registry \
  --username ${INSTALL_REGISTRY_USERNAME} --password ${INSTALL_REGISTRY_PASSWOR
D} \
  --server ${INSTALL_REGISTRY_HOSTNAME} \
  --export-to-all-namespaces --yes --namespace tap-install
```

6. Add the Tanzu Application Platform package repository to the cluster by running:

```
tanzu package repository add tanzu-tap-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}:$TAP_VERSION \
  --namespace tap-install
```

7. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

For example:

```
$ tanzu package repository get tanzu-tap-repository --namespace tap-install
- Retrieving repository tap...
NAME:          tanzu-tap-repository
VERSION:       16253001
REPOSITORY:    123456789012.dkr.ecr.us-west-2.amazonaws.com/tap-images
TAG:           1.5.12
STATUS:        Reconcile succeeded
REASON:
```

Note

The `VERSION` and `TAG` numbers differ from the earlier example if you are on Tanzu Application Platform v1.0.2 or earlier.

8. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com                   Application Accelerator
for VMware Tanzu                                    Used to create new projects a
nd configurations.
  api-portal.tanzu.vmware.com                         API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com                          API Auto Registration fo
r VMware Tanzu                                      A TAP component to automatica
lly register API exposing workloads as API entities

in TAP GUI.
  backend.appliveview.tanzu.vmware.com                Application Live View fo
r VMware Tanzu                                      App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com                       Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized

software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com          VMware Carbon Black for
Supply Chain Security Tools - Scan                  Default scan templates using
VMware Carbon Black
  cartographer.tanzu.vmware.com                       Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com                               Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com              Application Live View Co
nnector for VMware Tanzu                            App for discovering and regis
tering running apps
  controller.conventions.apps.tanzu.vmware.com        Convention Service for V
Mware Tanzu                                         Convention Service enables ap
p operators to consistently apply desired runtime

configurations to fleets of workloads.
  controller.source.apps.tanzu.vmware.com             Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.
  conventions.appliveview.tanzu.vmware.com            Application Live View Co
nventions for VMware Tanzu                          Application Live View convent
ion server
  developer-conventions.tanzu.vmware.com              Tanzu App Platform Devel
oper Conventions                                    Developer Conventions
  eventing.tanzu.vmware.com                           Eventing
Eventing is an event-driven architecture platform based on Knative Eventing
  external-secrets.apps.tanzu.vmware.com              External Secrets Operato
r                                                   External Secrets Operator is
a Kubernetes operator that integrates external

secret management systems.
  fluxcd.source.controller.tanzu.vmware.com           Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts

acquisition from external sources such as Git, Helm repositories and S3 bucket
s.
  grype.scanning.apps.tanzu.vmware.com                Grype for Supply Chain S
ecurity Tools - Scan                                Default scan templates using
Anchore Grype
  learningcenter.tanzu.vmware.com                     Learning Center for Tanz
u Application Platform                              Guided technical workshops
  metadata-store.apps.tanzu.vmware.com                Supply Chain Security To
ols - Store                                         Post SBoMs and query for imag
e, package, and vulnerability metadata.
  namespace-provisioner.apps.tanzu.vmware.com         Namespace Provisioner
Automatic Provisioning of Developer Namespaces.
  ootb-delivery-basic.tanzu.vmware.com                Tanzu App Platform Out o
```

```
f The Box Delivery Basic                              Out of The Box Delivery Basi
c.
  ootb-supply-chain-basic.tanzu.vmware.com              Tanzu App Platform Out o
f The Box Supply Chain Basic                          Out of The Box Supply Chain B
asic.
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning  Out of The Box Supply Chain w
ith Testing and Scanning.
  ootb-supply-chain-testing.tanzu.vmware.com            Tanzu App Platform Out o
f The Box Supply Chain with Testing                   Out of The Box Supply Chain w
ith Testing.
  ootb-templates.tanzu.vmware.com                       Tanzu App Platform Out o
f The Box Templates                                   Out of The Box Templates.
  policy.apps.tanzu.vmware.com                          Supply Chain Security To
ols - Policy Controller                               Policy Controller enables def
ining of a policy to restrict unsigned container

images.
  scanning.apps.tanzu.vmware.com                        Supply Chain Security To
ols - Scan                                            Scan for vulnerabilities and
enforce policies directly within Kubernetes native

Supply Chains.
  service-bindings.labs.vmware.com                      Service Bindings for Kub
ernetes                                               Service Bindings for Kubernet
es implements the Service Binding Specification.
  services-toolkit.tanzu.vmware.com                     Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and

connectivity of Service Resources (databases, message queues, DNS records,

etc.).
  snyk.scanning.apps.tanzu.vmware.com                   Snyk for Supply Chain Se
curity Tools - Scan                                   Default scan templates using
Snyk
  spring-boot-conventions.tanzu.vmware.com              Tanzu Spring Boot Conven
tions Server                                          Default Spring Boot conventio
n server.
  sso.apps.tanzu.vmware.com                             AppSSO
Application Single Sign-On for Tanzu
  tap-auth.tanzu.vmware.com                             Default roles for Tanzu
Application Platform                                  Default roles for Tanzu Appli
cation Platform
  tap-gui.tanzu.vmware.com                              Tanzu Application Platfo
rm GUI                                                web app graphical user interf
ace for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                        Telemetry Collector for
Tanzu Application Platform                            Tanzu Application Plaform Tel
emetry
  tap.tanzu.vmware.com                                  Tanzu Application Platfo
rm                                                    Package to install a set of T
AP components to get you started based on your use

case.
  tekton.tanzu.vmware.com                               Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.
  workshops.learningcenter.tanzu.vmware.com             Workshop Building Tutori
al                                                    Workshop Building Tutorial
```

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings. This is done by using the package manager installed by Tanzu Cluster Essentials.

For more information about profiles, see Components and installation profiles.

To prepare to install a profile:

1. List version information for the package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. Create a `tap-values.yaml` file by using the Full Profile (AWS), which contains the minimum configurations required to deploy Tanzu Application Platform on AWS. The sample values file contains the necessary defaults for:

   - The meta-package, or parent Tanzu Application Platform package.
   - Subordinate packages, or individual child packages.

   Keep the values file for future configuration use.

   > **Note**
   >
   > `tap-values.yaml` is set as a Kubernetes secret, which provides secure means to read credentials for Tanzu Application Platform components.

3. View possible configuration settings for your package

Full profile (AWS)

The following command generates the YAML file sample for the full-profile on AWS by using the ECR repositories you created earlier. The `profile:` field takes `full` as the default value, but you can also set it to `iterate`, `build`, `run`, or `view`. Refer to Install multicluster Tanzu Application Platform profiles for more information.

```
cat << EOF > tap-values.yaml
shared:
  ingress_domain: "INGRESS-DOMAIN"

ceip_policy_disclosed: true

#The above keys are minimum numbers of entries needed in tap-values.yaml to get a func
tioning TAP Full profile installation.

# Below are the keys which may have default values set, but can be overridden.

profile: full # Can take iterate, build, run, view.

supply_chain: basic # Can take testing, testing_scanning.

ootb_supply_chain_basic: # Based on supply_chain set above, can be changed to ootb_sup
ply_chain_testing, ootb_supply_chain_testing_scanning.
  registry:
    server: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com
    # The prefix of the ECR repository.  Workloads will need
    # two repositories created:
    #
    # tanzu-application-platform/<workloadname>-<namespace>
    # tanzu-application-platform/<workloadname>-<namespace>-bundle
    repository: tanzu-application-platform

contour:
  envoy:
    service:
      type: LoadBalancer # This is set by default, but can be overridden by setting a
different value.
```

```
buildservice:
  kp_default_repository: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/tap-bui
ld-service
  # Enable the build service k8s service account to bind to the AWS IAM Role
  kp_default_repository_aws_iam_role_arn: "arn:aws:iam::${AWS_ACCOUNT_ID}:role/tap-bui
ld-service"

ootb_templates:
  # Enable the config writer service to use cloud based iaas authentication
  # which are retrieved from the developer namespace service account by
  # default
  iaas_auth: true

tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml

metadata_store:
  ns_for_export_app_cert: "MY-DEV-NAMESPACE" # Verify this namespace is available with
in your cluster before initiating the Tanzu Application Platform installation.
  app_service_type: ClusterIP # Defaults to LoadBalancer. If shared.ingress_domain is
set earlier, this must be set to ClusterIP.

scanning:
  metadataStore:
    url: "" # Configuration is moved, so set this string to empty.

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.
EOF
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `kp_default_repository_aws_iam_role_arn` is the ARN that was created to write to the ECR repository for the build service. This value is generated by the script, but you can modify it manually.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the SCST - Store CA certificate to all namespaces.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports. See Kubernetes Grid documentation for more information about identifying the Entitlement Account Number.

For AWS, the default settings creates a classic LoadBalancer. To use the Network LoadBalancer instead of the classic LoadBalancer for ingress, add the following to your `tap-values.yaml`:

```
contour:
  infrastructure_provider: aws
  envoy:
    service:
      aws:
        LBType: nlb
```

(Optional) Configure your profile with full dependencies

When you install a profile that includes Tanzu Build Service, Tanzu Application Platform is installed with the `lite` set of dependencies. These dependencies consist of buildpacks and stacks required for application builds.

The `lite` set of dependencies do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  kp_default_repository: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/tap-bui
ld-service
  exclude_dependencies: true
```

After configuring `full` dependencies, you must install the dependencies after you have finished installing your Tanzu Application Platform package. See Install the full dependencies package for more information.

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v $TAP_VERSION --values-file
   tap-values.yaml -n tap-install
   ```

2. Verify the package install by running:

   ```
   tanzu package installed get tap -n tap-install
   ```

   This can take 5-10 minutes because it installs several packages on your cluster.

3. Verify that the necessary packages in the profile are installed by running:

   ```
   tanzu package installed list -A
   ```

4. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

After installing the Full profile on your cluster, you can install the Tanzu Developer Tools for VS Code Extension to help you develop against it. For instructions, see Install Tanzu Developer Tools for your VS Code.

Note

You can run the following command after reconfiguring the profile to reinstall the Tanzu Application Platform:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v $TAP_VERSION  --values-f
ile tap-values.yaml -n tap-install
```

Install the full dependencies package

If you configured `full` dependencies in your `tap-values.yaml` file in Configure your profile with full dependencies earlier, you must install the `full` dependencies package.

For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To install the `full` dependencies package:

1. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

   ```
   buildservice:
     kp_default_repository: ${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/
   tap-build-service
     exclude_dependencies: true
   ...
   ```

2. Get the latest version of the `buildservice` package by running:

   ```
   tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
   all
   ```

3. Create an ECR repository for Tanzu Build Service full dependencies by running:

   ```
   aws ecr create-repository --repository-name tbs-full-deps --region ${AWS_REGIO
   N}
   ```

4. Relocate the Tanzu Build Service full dependencies package repository by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
   ps-package-repo:VERSION \
     --to-repo ${INSTALL_REGISTRY_HOSTNAME}/tbs-full-deps
   ```

   Where `VERSION` is the version of the `buildservice` package you retrieved in the previous step.

5. Add the Tanzu Build Service full dependencies package repository by running:

   ```
   tanzu package repository add tbs-full-deps-repository \
     --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tbs-full-deps:VERSION \
     --namespace tap-install
   ```

   Where `VERSION` is the version of the `buildservice` package you retrieved earlier.

6. Install the full dependencies package by running:

   ```
   tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
   N -n tap-install
   ```

   Where `VERSION` is the version of the `buildservice` package you retrieved earlier.

Access Tanzu Application Platform GUI

To access Tanzu Application Platform GUI, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Application Platform GUI, see Access Tanzu Application Platform GUI.

You're now ready to start using Tanzu Application Platform GUI. Proceed to the Getting Started topic or the Tanzu Application Platform GUI - Catalog Operations topic.

Exclude packages from a Tanzu Application Platform profile

To exclude packages from a Tanzu Application Platform profile:

1. Find the full subordinate (child) package name:

   ```
   tanzu package available list --namespace tap-install
   ```

2. Update your `tap-values` file with a section listing the exclusions:

   ```
   profile: PROFILE-VALUE
   excluded_packages:
     - tap-gui.tanzu.vmware.com
     - service-bindings.lab.vmware.com
   ```

Important

If you exclude a package after performing a profile installation including that package, you cannot see the accurate package states immediately after running `tap package installed list -n tap-install`. Also, you can break package dependencies by removing a package. Allow 20 minutes to verify that all packages have reconciled correctly while troubleshooting.

Next steps

- (Optional) Install Individual Packages
- Set up developer namespaces to use your installed packages

View possible configuration settings for your package

To view possible configuration settings for a package, run:

```
tanzu package available get tap.tanzu.vmware.com/$TAP_VERSION --values-schema --namesp
ace tap-install
```

Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command

`tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNRS-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```
profile: full

# Shared configurations go under the shared key.
shared:
  ingress_domain: tap.example.com

# ...

# For example, CNRs specific values go under its name.
cnrs:
  provider: local

# For example, App Accelerator specific values go under its name.
accelerator:
  server:
    service_type: "ClusterIP"
```

Shared Keys define values that configure multiple packages. These keys are defined under the `shared` Top-level Key, as summarized in the following table:

Shared Key	Description	Optional
`ca_cert_data`	PEM-encoded certificate data to trust TLS connections with a private CA. This shared key is used by `convention_controller`, `scanning` and `source_controller`	Yes
`ingress_domain`	Domain name to be used in service routes and host names for instances of Tanzu Application Platform components.	Yes
`ingress_issuer`	A `cert-manager.io/v1/ClusterIssuer` for issuing TLS certificates to Tanzu Application Platform components. Default value: `tap-ingress-selfsigned`	Yes
`kubernetes_distribution`	Type of Kubernetes infrastructure being used. You can use this shared key in coordination with the `kubernetes_version` key. Supported value: `openshift`.	Yes
`kubernetes_version`	Kubernetes version. You can use this shared key independently or in coordination with the `kubernetes_distribution` key. Supported value: `1.24.x`, where `x` stands for the Kubernetes patch version.	Yes
`image_registry.project_path`	Project path in the container image registry server used for builder and application images.	Yes
`image_registry.username`	User name for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`image_registry.password`	Password for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`secret.name`	Secret name for the container image registry credentials of type `kubernetes.io/dockerconfigjson`. Mutually exclusive with `shared.image_registry.username/password`	Yes
`secret.namespace`	Secret namespace for the container image registry credentials. Mutually exclusive with `shared.image_registry.username/password`	Yes
`activateAppLiveViewSecureAccessControl`	Enable secure access connection between Application Live View components.	Yes

The following table summarizes the top-level keys used for package-specific configuration within your `tap-values.yaml`.

Package	Top-level Key
See table above.	`shared`
API Auto Registration	`api_auto_registration`
API portal	`api_portal`
Application Accelerator	`accelerator`
Application Live View	`appliveview`
Application Live View connector	`appliveview_connector`
Application Live View conventions	`appliveview-conventions`
Cartographer	`cartographer`
Cloud Native Runtimes	`cnrs`
Source Controller	`source_controller`
Supply Chain	`supply_chain`
Supply Chain Basic	`ootb_supply_chain_basic`
Supply Chain Testing	`ootb_supply_chain_testing`
Supply Chain Testing Scanning	`ootb_supply_chain_testing_scanning`
Supply Chain Security Tools - Scan	`scanning`
Supply Chain Security Tools - Scan (Grype Scanner)	`grype`
Supply Chain Security Tools - Store	`metadata_store`
Build Service	`buildservice`
Tanzu Application Platform GUI	`tap_gui`
Learning Center	`learningcenter`

For information about package-specific configuration, see Install individual packages.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service

- Install Application Live View
- Install Application Single Sign-On
- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform
- Install Developer Conventions
- Install Eventing
- Install Flux CD Source Controller
- Install Learning Center for Tanzu Application Platform
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit
- Install Source Controller
- Install Spring Boot conventions
- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Application Platform GUI
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package installed list --namespace tap-install
   \ Retrieving installed packages...
   NAME                    PACKAGE-NAME                                    PAC
   KAGE-VERSION  STATUS
   api-portal              api-portal.tanzu.vmware.com                     1.
   0.3           Reconcile succeeded
   app-accelerator         accelerator.apps.tanzu.vmware.com               1.
   ```

```
0.0            Reconcile succeeded
app-live-view            appliveview.tanzu.vmware.com                      1.
0.2            Reconcile succeeded
appliveview-conventions  build.appliveview.tanzu.vmware.com                1.
0.2            Reconcile succeeded
cartographer             cartographer.tanzu.vmware.com                     0.
1.0            Reconcile succeeded
cloud-native-runtimes    cnrs.tanzu.vmware.com                             1.
0.3            Reconcile succeeded
convention-controller    controller.conventions.apps.tanzu.vmware.com      0.
7.0            Reconcile succeeded
developer-conventions    developer-conventions.tanzu.vmware.com            0.
3.0-build.1    Reconcile succeeded
grype-scanner            grype.scanning.apps.tanzu.vmware.com              1.
0.0            Reconcile succeeded
image-policy-webhook     image-policy-webhook.signing.apps.tanzu.vmware.com 1.
1.2            Reconcile succeeded
metadata-store           metadata-store.apps.tanzu.vmware.com              1.
0.2            Reconcile succeeded
ootb-supply-chain-basic  ootb-supply-chain-basic.tanzu.vmware.com          0.
5.1            Reconcile succeeded
ootb-templates           ootb-templates.tanzu.vmware.com                   0.
5.1            Reconcile succeeded
scan-controller          scanning.apps.tanzu.vmware.com                    1.
0.0            Reconcile succeeded
service-bindings         service-bindings.labs.vmware.com                  0.
5.0            Reconcile succeeded
services-toolkit         services-toolkit.tanzu.vmware.com                 0.
8.0            Reconcile succeeded
source-controller        controller.source.apps.tanzu.vmware.com           0.
2.0            Reconcile succeeded
sso4k8s-install          sso.apps.tanzu.vmware.com                         1.
0.0-beta.2-31  Reconcile succeeded
tap-gui                  tap-gui.tanzu.vmware.com                          0.
3.0-rc.4       Reconcile succeeded
tekton-pipelines         tekton.tanzu.vmware.com                           0.3
0.0           Reconcile succeeded
tbs                      buildservice.tanzu.vmware.com                     1.
5.0            Reconcile succeeded
```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

You can choose either one of the following two approaches to create a `Workload` for your application by using the registry credentials specified, add credentials and Role-Based Access Control (RBAC) rules to the namespace that you plan to create the `Workload` in:

- Enable single user access.
- Enable additional users access with Kubernetes RBAC.

Enable single user access

Follow these steps to enable your current user to submit jobs to the Supply Chain:

1. (Optional) If the variable `AWS_ACCOUNT_ID environment` is not set during the installation process, export the AWS Account ID.

```
export AWS_ACCOUNT_ID=MY-AWS-ACCOUNT-ID
```

2. Add a service account to execute the supply chain and RBAC rules to authorize the service account to the developer namespace.

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
  annotations:
    eks.amazonaws.com/role-arn: "arn:aws:iam::${AWS_ACCOUNT_ID}:role/tap-worklo
ad"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-workload
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: workload
subjects:
  - kind: ServiceAccount
    name: default
EOF
```

Where `YOUR-NAMESPACE` is your developer namespace.

3. (Optional) If you haven't relocated the images to ECR, add a placeholder secret for gathering the credentials used for pulling container images.

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: e30K
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
  annotations:
    eks.amazonaws.com/role-arn: "arn:aws:iam::${AWS_ACCOUNT_ID}:role/tap-worklo
ad"
imagePullSecrets:
  - name: tap-registry
EOF
```

Where `YOUR-NAMESPACE` is your developer namespace.

Enable additional users access with Kubernetes RBAC

Follow these steps to enable additional users by using Kubernetes RBAC to submit jobs to the Supply Chain:

1. Enable single user access.

2. Choose either of the following options to give developers namespace-level access and view access to appropriate cluster-level resources:

    - **Option 1:** Use the Tanzu Application Platform RBAC CLI plug-in (beta).

      To use the `tanzu rbac` plug-in to grant `app-viewer` and `app-editor` roles to an identity provider group, run:

      ```
      tanzu rbac binding add -g GROUP-FOR-APP-VIEWER -n YOUR-NAMESPACE -r app-v
      iewer
      tanzu rbac binding add -g GROUP-FOR-APP-EDITOR -n YOUR-NAMESPACE -r app-e
      ditor
      ```

      Where:

        - `YOUR-NAMESPACE` is the name you give to the developer namespace.
        - `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
        - `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

      For more information about `tanzu rbac`, see Bind a user or group to a default role.

      VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

      Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrating Azure Active Directory.

    - **Option 2:** Use the native Kubernetes YAML.

      To apply the RBAC policy, run:

      ```
      cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
      apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
      metadata:
        name: dev-permit-app-viewer
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: app-viewer
      subjects:
        - kind: Group
          name: GROUP-FOR-APP-VIEWER
          apiGroup: rbac.authorization.k8s.io
      ---
      apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRoleBinding
      metadata:
        name: YOUR-NAMESPACE-permit-app-viewer
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: app-viewer-cluster-access
      subjects:
        - kind: Group
          name: GROUP-FOR-APP-VIEWER
          apiGroup: rbac.authorization.k8s.io
      ```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: YOUR-NAMESPACE-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
EOF
```

Where:

- `YOUR-NAMESPACE` is the name you give to the developer namespace.
- `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
- `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrating Azure Active Directory.

Rather than granting roles directly to individuals, VMware recommends using your identity provider's user groups system to grant access to a group of developers. For an example of how to set up Azure AD with your cluster, see Integrating Azure Active Directory.

3. (Optional) Log in as a non-admin user, such as a developer, to see the effects of RBAC after the bindings are applied.

Next steps

- Install Tanzu Developer Tools for your VS Code

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

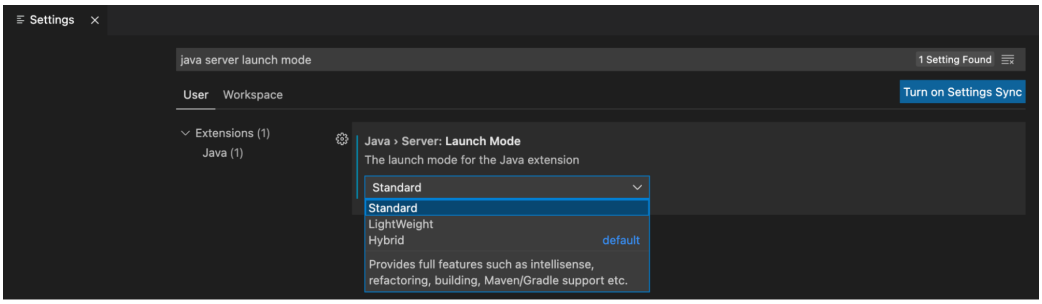
Install

To install the extension:

1. Sign in to VMware Tanzu Network and download Tanzu Developer Tools for Visual Studio Code.
2. Open VS Code.
3. Press cmd+shift+P to open the Command Palette and run `Extensions: Install from VSIX...`.

4. Select the extension file **tanzu-vscode-extension.vsix**.
5. If you do not have the following extensions, and they do not automatically install, install them from VS Code Marketplace:
   - Debugger for Java
   - Language Support for Java(™) by Red Hat
   - YAML
6. Ensure Language Support for Java is running in Standard Mode. You can configure it in the **Settings** menu by going to **Code** > **Preferences** > **Settings** under **Java > Server: Launch Mode**.

   When the JDK and Language Support for Java are configured correctly, you see that the integrated development environment creates a directory target where the code is compiled.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: For more information, see Integrating Live Hover by using Spring Boot Tools. Reload VS Code for this change to take effect.
    - **Source Image**: (Required) The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name.
    - **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
    - **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in `kubeconfig` is the default.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and select **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Install Tanzu Application Platform (Azure)

To install Tanzu Application Platform (commonly known as TAP) on Azure:

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create Azure Resources	Create Azure Resources
4.	Install Cluster Essentials for Tanzu	Deploy Cluster Essentials
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	Install the Tanzu Application Platform package and profiles
6.	(Optional) Install any additional packages that were not in the profile	Install individual packages
7.	Set up developer namespaces to use your installed packages	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE)	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform (Azure)

To install Tanzu Application Platform (commonly known as TAP) on Azure:

Step	Task	Link
1.	Review the prerequisites to ensure that you have set up everything required before beginning the installation	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create Azure Resources	Create Azure Resources
4.	Install Cluster Essentials for Tanzu	Deploy Cluster Essentials
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster	Install the Tanzu Application Platform package and profiles
6.	(Optional) Install any additional packages that were not in the profile	Install individual packages
7.	Set up developer namespaces to use your installed packages	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE)	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Create Azure Resources for Tanzu Application Platform

To install Tanzu Application Platform (commonly known as TAP) within the Azure ecosystem, you must create several Azure resources. Use this topic to learn how to create:

- An Azure Kubernetes Service (AKS) cluster to install Tanzu Application Platform.
- ACR repositories for the Tanzu Application Platform container images.

Creating these resources enables Tanzu Application Platform to use an IAM role bound to a Kubernetes service account for authentication, rather than the typical username and password stored in a Kubernetes secret strategy.

This is important when using ACR because authenticating to ACR is a two-step process:

1. Retrieve a token using your Azure credentials.
2. Use the token to authenticate to the registry.

To increase security, the token has a lifetime of 12 hours. This makes storing it as a secret for a service impractical because it must be refreshed every 12 hours.

Using an IAM role on a service account mitigates the need to retrieve the token because it is handled by credential helpers within the services.

Prerequisites

Before installing Tanzu Application Platform on Azure, you need:

- **An Azure subscription:**

You must create all of your resources within an Azure subscription and create an Azure free account.

- **Azure CLI:**

  To run CLI reference commands locally, you must install the Azure CLI. This topic uses Azure CLI to both query and configure resources in Azure, such as IAM roles. For more information, see Azure CLI documentation.

Create Azure Resource Group

1. Log in to Azure.

   ```
   az login
   az account set --subscription SUBSCRIPTION-NAME
   ```

2. Create a resource group with the `az group create` command.

   ```
   az group create --name myTAPResourceGroup --location eastus
   ```

Create an AKS cluster

To create an AKS cluster, you can run the az aks create command with the `--enable-addons monitoring` and `--enable-msi-auth-for-monitoring` parameter to enable Azure Monitor Container insights with managed identity authentication (preview).

The following example creates a cluster named `tap-on-azure` with one node and enables a system-assigned managed identity:

```
az aks create -g myTAPResourceGroup -n tap-on-azure --enable-managed-identity --node-c
ount 6 --enable-addons monitoring --enable-msi-auth-for-monitoring --generate-ssh-keys
--node-vm-size Standard_D4ds_v4 --kubernetes-version K8S-VERSION
```

Where `K8S-VERSION` is the compatible Kubernetes version that can be retrieved by running `az aks get-versions`.

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

Note

When you create an AKS cluster, a second resource group is automatically created to store the AKS resources. For more information, see Why are two resource groups created with AKS?

Connect to the AKS cluster

To manage a Kubernetes cluster, use the Kubernetes command-line client, kubectl. `kubectl` is already installed if you use Azure Cloud Shell.

1. Install `kubectl` locally by using the az aks install-cli command:

   ```
   az aks install-cli
   ```

2. Configure `kubectl` to connect to your Kubernetes cluster by using the az aks get-credentials command that:

- Downloads credentials and configures the Kubernetes CLI to use them.
- Uses `~/.kube/config`, the default location for the Kubernetes configuration file. You can specify a different location for your Kubernetes configuration file by using the `--file` argument.

```
az aks get-credentials --resource-group myTAPResourceGroup --name tap-on-azure
```

Create the container repositories

Azure Container Registry (ACR) does not require that the container repositories are already created. Repositories are created automatically when images are uploaded.

Enable registry admin account

To enable push and pull to your registries, you must enable the admin user account, which is created with each registry. Run the following command to enable the admin user account:

```
az acr update -n $REGISTRY_NAME --admin-enabled true
```

There are two passwords created for each admin user account per registry. To retrieve the passwords, run the following for each registry:

```
az acr credential show --name $REGISTRY_NAME --resource-group myTAPResourceGroup
```

Expect to see the following outputs:

```
{
  "passwords": [
    {
      "name": "password",
      "value": YOUR-PASSWORD
    },
    {
      "name": "password2",
      "value": YOUR-PASSWORD-2
    }
  ],
  "username": ""
}
```

Export the username and password by running:

```
export KP_REGISTRY_USERNAME=$REGISTRY_NAME
export KP_REGISTRY_PASSWORD=YOUR-PASSWORD
```

Next steps

- Install Tanzu Application Platform package and profiles on Azure

Install Tanzu Application Platform package and profiles on Azure

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) packages from your Tanzu Application Platform package repository on to Azure.

Before installing the packages, ensure you have:

- Completed the Prerequisites.
- Created Azure Resources.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.
- Installed Cluster Essentials for Tanzu.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

To relocate images from the VMware Tanzu Network registry to the ACR registry:

1. Set up environment variables for installation use by running:

   ```
   export AZURE_SP_APP_ID=MY-AZURE-APP-ID
   export AZURE_SP_TENANT=AZURE-TENANT
   export AZURE_SP_PASSWORD=AZURE-PASSWORD
   export AZURE_SUBSCRIPTION_ID=MY-AZURE-SUBSCRIPTION-ID
   export AZURE_ACCOUNT_ID=MY-AZURE-ACCOUNT-ID
   export AZURE_REGION=TARGET-AZURE-REGION
   export AKS_CLUSTER_NAME=tap-on-azure
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export IMGPKG_REGISTRY_HOSTNAME_1=$INSTALL_REGISTRY_HOSTNAME
   export IMGPKG_REGISTRY_USERNAME_1=$REGISTRY_NAME
   export IMGPKG_REGISTRY_PASSWORD_1=REGISTRY-PASSWORD
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REGISTRY_HOSTNAME=$REGISTRY_NAME.azurecr.io
   export INSTALL_REPO=tapimages
   ```

   Where:

   - `MY-AZURE-APP-ID` is the application ID you deploy Tanzu Application Platform in. Must be in UUID format.
   - `AZURE-TENANT` is the tenant you deploy Tanzu Application Platform in. Must be in UUID format.
   - `MY-AZURE-SUBSCRIPTION-ID` is the Azure subscription ID you deploy Tanzu Application Platform in. Must be in UUID format.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `TARGET-AZURE-REGION` is the region you deploy the Tanzu Application Platform to.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`

2. Install the Carvel tool imgpkg CLI.
3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy --concurrency 1 -b ${IMGPKG_REGISTRY_HOSTNAME_0}/tanzu-application-platform/tap-packages:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}
   ```

4. Create a namespace called `tap-install` for deploying any component packages by running:

   ```
   kubectl create ns tap-install
   ```

   This namespace keeps the objects grouped together logically.

5. Export the VMware Tanzu Network registry by running:

   ```
   export INSTALL_REPO=tanzu-application-platform/tap-packages
   ```

6. Create registry secret for the VMware Tanzu Network registry by running:

   ```
   tanzu secret registry add tap-registry \
   --username ${INSTALL_REGISTRY_USERNAME} --password ${INSTALL_REGISTRY_PASSWORD}
   \
   --server ${INSTALL_REGISTRY_HOSTNAME} \
   --export-to-all-namespaces --yes --namespace tap-install
   tanzu secret registry list --namespace tap-install
   ```

7. Add the Tanzu Application Platform package repository to the cluster by running:

   ```
   tanzu package repository add tanzu-tap-repository \
     --url ${INSTALL_REGISTRY_HOSTNAME}/full-deps-package-repo
     --namespace tap-install
   ```

8. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

   ```
   tanzu package repository get tanzu-tap-repository --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package repository get tanzu-tap-repository --namespace tap-install
   - Retrieving repository tap...
   NAME:          tanzu-tap-repository
   VERSION:       16253001
   REPOSITORY:    123456789012.dkr.acr.us-east.azure.com/tap-images
   TAG:           1.5.12
   STATUS:        Reconcile succeeded
   REASON:
   ```

   Note

   The `VERSION` and `TAG` numbers differ from the earlier example if you are on Tanzu Application Platform v1.0.2 or earlier.

9. List the available packages by running:

   ```
   tanzu package available list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list --namespace tap-install
   / Retrieving available packages...
     NAME                                                DISPLAY-NAME
   SHORT-DESCRIPTION
     accelerator.apps.tanzu.vmware.com                   Application Accelerator
   for VMware Tanzu                                  Used to create new projects a
   nd configurations.
   ```

```
  api-portal.tanzu.vmware.com                              API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com                               API Auto Registration fo
r VMware Tanzu                                        A TAP component to automatica
lly register API exposing workloads as API entities

in TAP GUI.
  backend.appliveview.tanzu.vmware.com                     Application Live View fo
r VMware Tanzu                                        App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com                            Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized

software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com               VMware Carbon Black for
Supply Chain Security Tools - Scan                    Default scan templates using
VMware Carbon Black
  cartographer.tanzu.vmware.com                            Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com                                    Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com                   Application Live View Co
nnector for VMware Tanzu                              App for discovering and regis
tering running apps
  controller.source.apps.tanzu.vmware.com                  Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.
  conventions.appliveview.tanzu.vmware.com                 Application Live View Co
nventions for VMware Tanzu                            Application Live View convent
ion server
  developer-conventions.tanzu.vmware.com                   Tanzu App Platform Devel
oper Conventions                                      Developer Conventions
  eventing.tanzu.vmware.com                                Eventing
Eventing is an event-driven architecture platform based on Knative Eventing
  external-secrets.apps.tanzu.vmware.com                   External Secrets Operato
r                                                     External Secrets Operator is
a Kubernetes operator that integrates external

secret management systems.
  fluxcd.source.controller.tanzu.vmware.com                Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts

acquisition from external sources such as Git, Helm repositories and S3 bucket
s.
  grype.scanning.apps.tanzu.vmware.com                     Grype for Supply Chain S
ecurity Tools - Scan                                  Default scan templates using
Anchore Grype
  learningcenter.tanzu.vmware.com                          Learning Center for Tanz
u Application Platform                                Guided technical workshops
  metadata-store.apps.tanzu.vmware.com                     Supply Chain Security To
ols - Store                                           Post SBoMs and query for imag
e, package, and vulnerability metadata.
  namespace-provisioner.apps.tanzu.vmware.com              Namespace Provisioner
Automatic Provisioning of Developer Namespaces.
  ootb-delivery-basic.tanzu.vmware.com                     Tanzu App Platform Out o
f The Box Delivery Basic                              Out of The Box Delivery Basi
c.
  ootb-supply-chain-basic.tanzu.vmware.com                 Tanzu App Platform Out o
f The Box Supply Chain Basic                          Out of The Box Supply Chain B
asic.
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning  Out of The Box Supply Chain w
ith Testing and Scanning.
  ootb-supply-chain-testing.tanzu.vmware.com               Tanzu App Platform Out o
f The Box Supply Chain with Testing                   Out of The Box Supply Chain w
ith Testing.
  ootb-templates.tanzu.vmware.com                          Tanzu App Platform Out o
```

```
f The Box Templates                                Out of The Box Templates.
  policy.apps.tanzu.vmware.com                          Supply Chain Security To
ols - Policy Controller                            Policy Controller enables def
ining of a policy to restrict unsigned container

images.
  scanning.apps.tanzu.vmware.com                        Supply Chain Security To
ols - Scan                                         Scan for vulnerabilities and
enforce policies directly within Kubernetes native

Supply Chains.
  service-bindings.labs.vmware.com                      Service Bindings for Kub
ernetes                                            Service Bindings for Kubernet
es implements the Service Binding Specification.
  services-toolkit.tanzu.vmware.com                     Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and

connectivity of Service Resources (databases, message queues, DNS records,

etc.).
  snyk.scanning.apps.tanzu.vmware.com                   Snyk for Supply Chain Se
curity Tools - Scan                                Default scan templates using
Snyk
  spring-boot-conventions.tanzu.vmware.com              Tanzu Spring Boot Conven
tions Server                                       Default Spring Boot conventio
n server.
  sso.apps.tanzu.vmware.com                             AppSSO
Application Single Sign-On for Tanzu
  tap-auth.tanzu.vmware.com                             Default roles for Tanzu
Application Platform                               Default roles for Tanzu Appli
cation Platform
  tap-gui.tanzu.vmware.com                              Tanzu Application Platfo
rm GUI                                             web app graphical user interf
ace for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                        Telemetry Collector for
Tanzu Application Platform                         Tanzu Application Platform Te
lemetry
  tap.tanzu.vmware.com                                  Tanzu Application Platfo
rm                                                 Package to install a set of T
AP components to get you started based on your use

case.
  tekton.tanzu.vmware.com                               Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.
  workshops.learningcenter.tanzu.vmware.com             Workshop Building Tutori
al                                                 Workshop Building Tutorial
```

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings by using the package manager installed by Tanzu Cluster Essentials. For more information about profiles, see Components and installation profiles.

To create a registry secret and add it to a developer namespace:

```
export KP_REGISTRY_USERNAME=YOUR-USERNAME
export KP_REGISTRY_PASSWORD=YOUR-PASSWORD
export KP_REGISTRY_HOSTNAME=YOUR-HOSTNAME

echo $KP_REGISTRY_USERNAME
echo $KP_REGISTRY_PASSWORD
echo $KP_REGISTRY_HOSTNAME

docker login $KP_REGISTRY_HOSTNAME -u $KP_REGISTRY_USERNAME -p $KP_REGISTRY_PASSWORD
```

```
export YOUR_NAMESPACE=mydev-ns

echo $YOUR_NAMESPACE

kubectl create ns $YOUR_NAMESPACE

tanzu secret registry add registry-credentials --server $KP_REGISTRY_HOSTNAME --userna
me $KP_REGISTRY_USERNAME --password $KP_REGISTRY_PASSWORD --namespace $YOUR_NAMESPACE

kubectl get secret registry-credentials  -o jsonpath='{.data.\.dockerconfigjson}'  -n
$YOUR_NAMESPACE| base64 --decode
```

To prepare to install a profile:

1. List version information for the package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. Create a `tap-values.yaml` file by using the Full Profile (Azure), which contains the minimum configurations required to deploy Tanzu Application Platform on Azure. The sample values file contains the necessary defaults for:

   - The meta-package, or parent Tanzu Application Platform package.
   - Subordinate packages, or individual child packages.

   Keep the values file for future configuration use.

   > **Note**
   >
   > `tap-values.yaml` is set as a Kubernetes secret, which provides secure means to read credentials for Tanzu Application Platform components.

3. View possible configuration settings for your package

Full profile (Azure)

The following is the YAML file sample for the full-profile on Azure by using the ACR repositories you created earlier. The `profile:` field takes `full` as the default value, but you can also set it to `iterate`, `build`, `run`, or `view`. See Install multicluster Tanzu Application Platform profiles for more information.

```
cat << EOF > tap-values.yaml
ceip_policy_disclosed: true
profile: full # Can take iterate, build, run, view.

supply_chain: basic # Can take testing, testing_scanning.

ootb_templates:
  iaas_auth: true

ootb_supply_chain_basic:
  registry:
    server: ${KP_REGISTRY_HOSTNAME}
    repository: ${INSTALL_REPO}
  gitops:
    ssh_secret: ""

contour:
  envoy:
    service:
```

```
      type: LoadBalancer

buildservice:
  kp_default_repository: ${KP_REGISTRY_HOSTNAME}
  kp_default_repository_secret:
    name: registry-credentials
    namespace: "MY-DEV-NAMESPACE"
  enable_automatic_dependency_updates: false

learningcenter:
  ingressDomain: learning-center.tap.com

ootb_delivery_basic:
  service_account: default

tap_gui:
  ingressEnabled: true
  ingressDomain: tap.com
  app_config:
    supplyChain:
      enablePlugin: true
    auth:
      allowGuestAccess: true
    backend:
      baseUrl: http://tap-gui.tap.com
      cors:
        origin: http://tap-gui.tap.com
    app:
      baseUrl: http://tap-gui.tap.com

metadata_store:
  ingressEnabled: true
  ingressDomain: "INGRESS-DOMAIN"
  app_service_type: ClusterIP
  ns_for_export_app_cert: "MY-DEV-NAMESPACE"

scanning:
  metadataStore:
    url: "" # Configuration is moved, so set this string to empty.

accelerator:
  server:
    service_type: "ClusterIP"

cnrs:
  domain_name: tap.com
EOF
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, you can use a Backstage-compliant catalog that was built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the SCST - Store CA certificate to all namespaces.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.

(Optional) Additional Build Service configurations

The following tasks are optional during the Tanzu Application Platform installation process:

- (Optional) Configure your profile with full dependencies
- (Optional) Configure your profile with the Jammy stack only

(Optional) Configure your profile with full dependencies

When you install a profile that includes Tanzu Build Service, Tanzu Application Platform is installed with the `lite` set of dependencies. These dependencies consist of buildpacks and stacks required for application builds.

The `lite` set of dependencies do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section by default,
but can be overridden by setting a different value.
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  exclude_dependencies: true
```

After configuring `full` dependencies, you must install the dependencies after you have finished installing your Tanzu Application Platform package. See Install the full dependencies package for more information.

(Optional) Configure your profile with the Jammy stack only

Tanzu Application Platform v1.5.0 supports building applications with both the Ubuntu v22.04 (Jammy) and v18.04 (Bionic) stack. For more information, see Bionic and Jammy stacks.

To install Tanzu Application Platform with Jammy as the only available stack, include the `stack_configuration: jammy-only` field under the `buildservice:` section in `tap-values.yaml`.

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v $TAP_VERSION --values-file
   tap-values.yaml -n tap-install
   ```

2. Verify the package install by running:

   ```
   tanzu package installed get tap -n tap-install
   ```

   This can take 5-10 minutes because it installs several packages on your cluster.

3. Verify that the necessary packages in the profile are installed by running:

   ```
   tanzu package installed list -A
   ```

4. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

After installing the Full profile on your cluster, you can install the Tanzu Developer Tools for VS Code Extension to help you develop against it. For more information, see Install Tanzu Developer Tools for your VS Code.

Install the full dependencies package

If you configured `full` dependencies in your `tap-values.yaml` file in Configure your profile with full dependencies earlier, you must install the `full` dependencies package.

For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To install the `full` dependencies package:

1. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

   ```
   buildservice:
     kp_default_repository: ${KP_REGISTRY_HOSTNAME}.azurecr.io/{$REPOSITORY_NAME}
     exclude_dependencies: true
   ...
   ```

2. Get the latest version of the `buildservice` package by running:

   ```
   tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
   all
   ```

3. Relocate the Tanzu Build Service full dependencies package repository by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
   ps-package-repo:VERSION \
     --to-repo ${INSTALL_REGISTRY_HOSTNAME}/tbs-full-deps
   ```

   Where `VERSION` is the version of the `buildservice` package you retrieved in the previous step.

4. Add the Tanzu Build Service full dependencies package repository by running:

   ```
   tanzu package repository add tbs-full-deps-repository \
     --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tbs-full-deps:VERSION \
     --namespace tap-install
   ```

   Where `VERSION` is the version of the `tap` package you retrieved earlier.

5. Install the full dependencies package by running:

   ```
   tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
   N -n tap-install
   ```

   Where `VERSION` is the version of the `buildservice` package you retrieved earlier.

Access Tanzu Application Platform GUI

To access Tanzu Application Platform GUI, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Application Platform GUI, see Access Tanzu Application Platform GUI.

You're now ready to start using Tanzu Application Platform GUI. Proceed to the Getting Started topic or the Tanzu Application Platform GUI - Catalog Operations topic.

Next steps

- (Optional) Install Individual Packages
- Set up developer namespaces to use your installed packages

View possible configuration settings for your package

To view possible configuration settings for a package, run:

```
tanzu package available get tap.tanzu.vmware.com/$TAP_VERSION --values-schema --namesp
ace tap-install
```

Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNRS-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```
profile: full

# Shared configurations go under the shared key.
shared:
  ingress_domain: tap.example.com

# ...

# For example, CNRs specific values go under its name.
cnrs:
  provider: local

# For example, App Accelerator specific values go under its name.
accelerator:
  server:
    service_type: "ClusterIP"
```

Shared Keys define values that configure multiple packages. These keys are defined under the `shared` Top-level Key, as summarized in the following table:

Shared Key	Description	Optional
`ca_cert_data`	PEM-encoded certificate data to trust TLS connections with a private CA. This shared key is used by `convention_controller`, `scanning` and `source_controller`	Yes
`ingress_domain`	Domain name to be used in service routes and host names for instances of Tanzu Application Platform components.	Yes
`ingress_issuer`	A `cert-manager.io/v1/ClusterIssuer` for issuing TLS certificates to Tanzu Application Platform components. Default value: `tap-ingress-selfsigned`	Yes

Shared Key	Description	Optional
`kubernetes_distribution`	Type of Kubernetes infrastructure being used. You can use this shared key in coordination with the `kubernetes_version` key. Supported value: `openshift`.	Yes
`kubernetes_version`	Kubernetes version. You can use this shared key independently or in coordination with the `kubernetes_distribution` key. Supported value: `1.24.x`, where `x` stands for the Kubernetes patch version.	Yes
`image_registry.project_path`	Project path in the container image registry server used for builder and application images.	Yes
`image_registry.username`	User name for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`image_registry.password`	Password for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`secret.name`	Secret name for the container image registry credentials of type `kubernetes.io/dockerconfigjson`. Mutually exclusive with `shared.image_registry.username/password`	Yes
`secret.namespace`	Secret namespace for the container image registry credentials. Mutually exclusive with `shared.image_registry.username/password`	Yes
`activateAppLiveViewSecureAccessControl`	Enable secure access connection between Application Live View components.	Yes

The following table summarizes the top-level keys used for package-specific configuration within your `tap-values.yaml`.

Package	Top-level Key
See table above.	`shared`
API Auto Registration	`api_auto_registration`
API portal	`api_portal`
Application Accelerator	`accelerator`
Application Live View	`appliveview`
Application Live View connector	`appliveview_connector`
Application Live View conventions	`appliveview-conventions`
Cartographer	`cartographer`
Cloud Native Runtimes	`cnrs`
Source Controller	`source_controller`
Supply Chain	`supply_chain`
Supply Chain Basic	`ootb_supply_chain_basic`
Supply Chain Testing	`ootb_supply_chain_testing`
Supply Chain Testing Scanning	`ootb_supply_chain_testing_scanning`
Supply Chain Security Tools - Scan	`scanning`
Supply Chain Security Tools - Scan (Grype Scanner)	`grype`
Supply Chain Security Tools - Store	`metadata_store`
Build Service	`buildservice`

Package	Top-level Key
Tanzu Application Platform GUI	`tap_gui`
Learning Center	`learningcenter`

For information about package-specific configuration, see Install individual packages.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service
- Install Application Live View
- Install Application Single Sign-On
- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform
- Install Developer Conventions
- Install Eventing
- Install Flux CD Source Controller
- Install Learning Center for Tanzu Application Platform
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit
- Install Source Controller
- Install Spring Boot conventions

- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Application Platform GUI
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package installed list --namespace tap-install
   \ Retrieving installed packages...
   NAME                     PACKAGE-NAME                                      PAC
   KAGE-VERSION  STATUS
   api-portal               api-portal.tanzu.vmware.com                       1.
   0.3           Reconcile succeeded
   app-accelerator          accelerator.apps.tanzu.vmware.com                 1.
   0.0           Reconcile succeeded
   app-live-view            appliveview.tanzu.vmware.com                      1.
   0.2           Reconcile succeeded
   appliveview-conventions  build.appliveview.tanzu.vmware.com                1.
   0.2           Reconcile succeeded
   cartographer             cartographer.tanzu.vmware.com                     0.
   1.0           Reconcile succeeded
   cloud-native-runtimes    cnrs.tanzu.vmware.com                             1.
   0.3           Reconcile succeeded
   convention-controller    controller.conventions.apps.tanzu.vmware.com      0.
   7.0           Reconcile succeeded
   developer-conventions    developer-conventions.tanzu.vmware.com            0.
   3.0-build.1   Reconcile succeeded
   grype-scanner            grype.scanning.apps.tanzu.vmware.com              1.
   0.0           Reconcile succeeded
   image-policy-webhook     image-policy-webhook.signing.apps.tanzu.vmware.com 1.
   1.2           Reconcile succeeded
   metadata-store           metadata-store.apps.tanzu.vmware.com              1.
   0.2           Reconcile succeeded
   ootb-supply-chain-basic  ootb-supply-chain-basic.tanzu.vmware.com          0.
   5.1           Reconcile succeeded
   ootb-templates           ootb-templates.tanzu.vmware.com                   0.
   5.1           Reconcile succeeded
   scan-controller          scanning.apps.tanzu.vmware.com                    1.
   0.0           Reconcile succeeded
   service-bindings         service-bindings.labs.vmware.com                  0.
   5.0           Reconcile succeeded
   services-toolkit         services-toolkit.tanzu.vmware.com                 0.
   8.0           Reconcile succeeded
   source-controller        controller.source.apps.tanzu.vmware.com           0.
   2.0           Reconcile succeeded
   sso4k8s-install          sso.apps.tanzu.vmware.com                         1.
   0.0-beta.2-31  Reconcile succeeded
   ```

```
tap-gui                   tap-gui.tanzu.vmware.com                              0.
3.0-rc.4       Reconcile succeeded
tekton-pipelines          tekton.tanzu.vmware.com                               0.3
0.0            Reconcile succeeded
tbs                       buildservice.tanzu.vmware.com                         1.
5.0             Reconcile succeeded
```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

This topic tells you how to set up developer namespaces by using the legacy manual process. For more information about how to automatically set up your developer namespaces, see Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

You can choose either one of the following two approaches to create a `Workload` for your application by using the registry credentials specified, add credentials and Role-Based Access Control (RBAC) rules to the namespace that you plan to create the `Workload` in:

- Enable single user access.
- Enable additional users access with Kubernetes RBAC.

Enable single user access

Run the following command to add secrets, a service account to execute the supply chain, and RBAC rules to authorize the service account to the developer namespace:

```
cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: tap-registry
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: e30K

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
imagePullSecrets:
```

```
  - name: registry-credentials
  - name: tap-registry

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: default
rules:
- apiGroups: [source.toolkit.fluxcd.io]
  resources: [gitrepositories]
  verbs: ['*']
- apiGroups: [source.apps.tanzu.vmware.com]
  resources: [imagerepositories]
  verbs: ['*']
- apiGroups: [carto.run]
  resources: [deliverables, runnables]
  verbs: ['*']
- apiGroups: [kpack.io]
  resources: [images]
  verbs: ['*']
- apiGroups: [conventions.apps.tanzu.vmware.com]
  resources: [podintents]
  verbs: ['*']
- apiGroups: [""]
  resources: ['configmaps']
  verbs: ['*']
- apiGroups: [""]
  resources: ['pods']
  verbs: ['list']
- apiGroups: [tekton.dev]
  resources: [taskruns, pipelineruns]
  verbs: ['*']
- apiGroups: [tekton.dev]
  resources: [pipelines]
  verbs: ['list']
- apiGroups: [kappctrl.k14s.io]
  resources: [apps]
  verbs: ['*']
- apiGroups: [serving.knative.dev]
  resources: ['services']
  verbs: ['*']
- apiGroups: [servicebinding.io]
  resources: ['servicebindings']
  verbs: ['*']
- apiGroups: [services.apps.tanzu.vmware.com]
  resources: ['resourceclaims']
  verbs: ['*']
- apiGroups: [scanning.apps.tanzu.vmware.com]
  resources: ['imagescans', 'sourcescans']
  verbs: ['*']

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: default
subjects:
  - kind: ServiceAccount
    name: default
```

Where `YOUR-NAMESPACE` is your developer namespace.

Enable additional users access with Kubernetes RBAC

Follow these steps to enable additional users by using Kubernetes RBAC to submit jobs to the Supply Chain:

1. Enable single user access.
2. Choose either of the following options to give developers namespace-level access and view access to appropriate cluster-level resources:
    - **Option 1:** Use the Tanzu Application Platform RBAC CLI plug-in (beta).

      To use the `tanzu rbac` plug-in to grant `app-viewer` and `app-editor` roles to an identity provider group, run:

      ```
      tanzu rbac binding add -g GROUP-FOR-APP-VIEWER -n YOUR-NAMESPACE -r app-v
      iewer
      tanzu rbac binding add -g GROUP-FOR-APP-EDITOR -n YOUR-NAMESPACE -r app-e
      ditor
      ```

      Where:

      - `YOUR-NAMESPACE` is the name you give to the developer namespace.
      - `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
      - `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

      For more information about `tanzu rbac`, see Bind a user or group to a default role.

      VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

      Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrating Azure Active Directory.

    - **Option 2:** Use the native Kubernetes YAML.

      To apply the RBAC policy, run:

      ```
      cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
      apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
      metadata:
        name: dev-permit-app-viewer
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: app-viewer
      subjects:
        - kind: Group
          name: GROUP-FOR-APP-VIEWER
          apiGroup: rbac.authorization.k8s.io
      ---
      apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRoleBinding
      metadata:
        name: YOUR-NAMESPACE-permit-app-viewer
      roleRef:
      ```

```
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-viewer-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-VIEWER
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: YOUR-NAMESPACE-permit-app-editor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: app-editor-cluster-access
subjects:
  - kind: Group
    name: GROUP-FOR-APP-EDITOR
    apiGroup: rbac.authorization.k8s.io
EOF
```

Where:

- `YOUR-NAMESPACE` is the name you give to the developer namespace.
- `GROUP-FOR-APP-VIEWER` is the user group from the upstream identity provider that requires access to `app-viewer` resources on the current namespace and cluster.
- `GROUP-FOR-APP-EDITOR` is the user group from the upstream identity provider that requires access to `app-editor` resources on the current namespace and cluster.

VMware recommends creating a user group in your identity provider's grouping system for each developer namespace and then adding the users accordingly.

Depending on your identity provider, you might need to take further action to federate user groups appropriately with your cluster. For an example of how to set up Azure Active Directory (AD) with your cluster, see Integrating Azure Active Directory.

Rather than granting roles directly to individuals, VMware recommends using your identity provider's user groups system to grant access to a group of developers. For an example of how to set up Azure AD with your cluster, see Integrating Azure Active Directory.

3. (Optional) Log in as a non-admin user, such as a developer, to see the effects of RBAC after the bindings are applied.

Next steps

- Install Tanzu Developer Tools for your VS Code

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

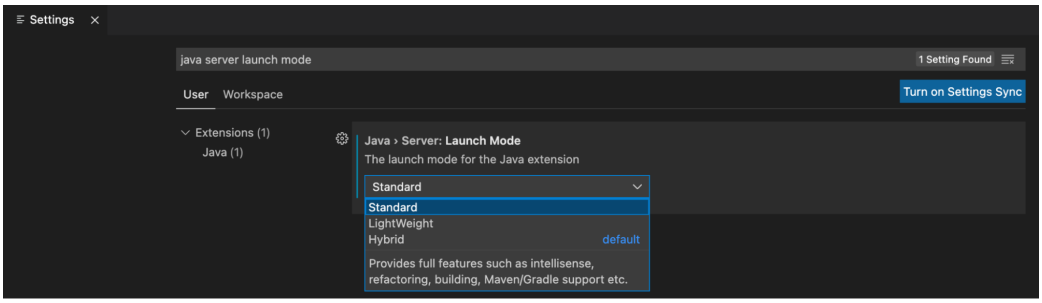
Install

To install the extension:

1. Sign in to VMware Tanzu Network and download Tanzu Developer Tools for Visual Studio Code.
2. Open VS Code.
3. Press cmd+shift+P to open the Command Palette and run `Extensions: Install from VSIX...`.

4. Select the extension file **tanzu-vscode-extension.vsix**.
5. If you do not have the following extensions, and they do not automatically install, install them from VS Code Marketplace:
   - Debugger for Java
   - Language Support for Java(™) by Red Hat
   - YAML
6. Ensure Language Support for Java is running in Standard Mode. You can configure it in the **Settings** menu by going to **Code** > **Preferences** > **Settings** under **Java > Server: Launch Mode**.



When the JDK and Language Support for Java are configured correctly, you see that the integrated development environment creates a directory target where the code is compiled.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: For more information, see Integrating Live Hover by using Spring Boot Tools. Reload VS Code for this change to take effect.
    - **Source Image**: (Required) The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name.
    - **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
    - **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in `kubeconfig` is the default.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and select **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Install Tanzu Application Platform (OpenShift)

To install Tanzu Application Platform (commonly known as TAP) on your OpenShift clusters with internet access:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install the Tanzu Application Platform package and profiles
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on to your OpenShift clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform (OpenShift)

To install Tanzu Application Platform (commonly known as TAP) on your OpenShift clusters with internet access:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu.	Deploy Cluster Essentials
4.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install the Tanzu Application Platform package and profiles
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

After installing Tanzu Application Platform on to your OpenShift clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform on your OpenShift clusters

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) packages on your OpenShift clusters.

Before installing the packages, ensure you have:

- Completed the Prerequisites.
- Configured and verified the cluster.

- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform will depend on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following documentation for a registry to learn how to set it up:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

   ```
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
   export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
   export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_USERNAME="${IMGPKG_REGISTRY_USERNAME_1}"
   export INSTALL_REGISTRY_PASSWORD="${IMGPKG_REGISTRY_PASSWORD_1}"
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own container registry.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.
   - `TARGET-REPOSITORY` is your target repository, a folder/repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.
2. Install the Carvel tool imgpkg CLI.
3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages
   ```
4. Create a namespace called `tap-install` for deploying any component packages by running:

   ```
   kubectl create ns tap-install
   ```

This namespace keeps the objects grouped together logically.

5. Create a registry secret by running:

```
tanzu secret registry add tap-registry \
  --username ${INSTALL_REGISTRY_USERNAME} --password ${INSTALL_REGISTRY_PASSWOR
D} \
  --server ${INSTALL_REGISTRY_HOSTNAME} \
  --export-to-all-namespaces --yes --namespace tap-install
```

6. (Optional) Create a registry secret for your writable image repository used for:
    - Tanzu Build Service Dependencies
    - Workloads when using the `shared.image_registry` key

```
tanzu secret registry add image-registry-creds \
  --server "${REGISTRY_HOSTNAME}" \
  --username "${REGISTRY_USERNAME}" \
  --password "${REGISTRY_PASSWORD}" \
  --namespace tap-install
```

Where:

- `REGISTRY_HOSTNAME` is the host name for the registry that contains your writable repository. Examples:
    - Harbor has the form `--server "my-harbor.io"`.
    - Docker Hub has the form `--server "index.docker.io"`.
    - Google Cloud Registry has the form `--server "gcr.io"`.
- `REGISTRY_USERNAME` and `REGISTRY_PASSWORD` are the user name and password for the user that can write to the repository used in the following step. For Google Cloud Registry, use `_json_key` as the user name and the contents of the service account JSON file for the password.

Note

If using the same repository as `tap-registry`, you can skip this step and use the `tap-registry` secret in your `tap-values.yaml` instead of `image-registry-creds`.

7. Add the Tanzu Application Platform package repository to the cluster by running:

```
tanzu package repository add tanzu-tap-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages:$TAP_VERSION
\
  --namespace tap-install
```

8. Get the status of the Tanzu Application Platform package repository, and ensure the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

For example:

```
$ tanzu package repository get tanzu-tap-repository --namespace tap-install
- Retrieving repository tap...
NAME:          tanzu-tap-repository
VERSION:       16253001
```

```
REPOSITORY:    tapmdc.azurecr.io/mdc/1.4.0/tap-packages
TAG:           1.5.12
STATUS:        Reconcile succeeded
REASON:
```

Note

The `VERSION` and `TAG` numbers differ from the earlier example if you are on Tanzu Application Platform v1.0.2 or earlier.

9. List the available packages by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  accelerator.apps.tanzu.vmware.com                   Application Accelerator
for VMware Tanzu                                    Used to create new projects a
nd configurations.
  api-portal.tanzu.vmware.com                         API portal
A unified user interface for API discovery and exploration at scale.
  apis.apps.tanzu.vmware.com                          API Auto Registration fo
r VMware Tanzu                                      A TAP component to automatica
lly register API exposing workloads as API entities

in TAP GUI.
  backend.appliveview.tanzu.vmware.com                Application Live View fo
r VMware Tanzu                                      App for monitoring and troubl
eshooting running apps
  buildservice.tanzu.vmware.com                       Tanzu Build Service
Tanzu Build Service enables the building and automation of containerized

software workflows securely and at scale.
  carbonblack.scanning.apps.tanzu.vmware.com          VMware Carbon Black for
Supply Chain Security Tools - Scan                  Default scan templates using
VMware Carbon Black
  cartographer.tanzu.vmware.com                       Cartographer
Kubernetes native Supply Chain Choreographer.
  cnrs.tanzu.vmware.com                               Cloud Native Runtimes
Cloud Native Runtimes is a serverless runtime based on Knative
  connector.appliveview.tanzu.vmware.com              Application Live View Co
nnector for VMware Tanzu                            App for discovering and regis
tering running apps
  controller.source.apps.tanzu.vmware.com             Tanzu Source Controller
Tanzu Source Controller enables workload create/update from source code.
  conventions.appliveview.tanzu.vmware.com            Application Live View Co
nventions for VMware Tanzu                          Application Live View convent
ion server
  developer-conventions.tanzu.vmware.com              Tanzu App Platform Devel
oper Conventions                                    Developer Conventions
  eventing.tanzu.vmware.com                           Eventing
Eventing is an event-driven architecture platform based on Knative Eventing
  external-secrets.apps.tanzu.vmware.com              External Secrets Operato
r                                                   External Secrets Operator is
a Kubernetes operator that integrates external

secret management systems.
  fluxcd.source.controller.tanzu.vmware.com           Flux Source Controller
The source-controller is a Kubernetes operator, specialised in artifacts
```

```
acquisition from external sources such as Git, Helm repositories and S3 bucket
s.
  grype.scanning.apps.tanzu.vmware.com                Grype for Supply Chain S
ecurity Tools - Scan                                Default scan templates using
Anchore Grype
  learningcenter.tanzu.vmware.com                     Learning Center for Tanz
u Application Platform                              Guided technical workshops
  metadata-store.apps.tanzu.vmware.com                Supply Chain Security To
ols - Store                                         Post SBoMs and query for imag
e, package, and vulnerability metadata.
  namespace-provisioner.apps.tanzu.vmware.com         Namespace Provisioner
Automatic Provisioning of Developer Namespaces.
  ootb-delivery-basic.tanzu.vmware.com                Tanzu App Platform Out o
f The Box Delivery Basic                            Out of The Box Delivery Basi
c.
  ootb-supply-chain-basic.tanzu.vmware.com            Tanzu App Platform Out o
f The Box Supply Chain Basic                        Out of The Box Supply Chain B
asic.
  ootb-supply-chain-testing-scanning.tanzu.vmware.com  Tanzu App Platform Out o
f The Box Supply Chain with Testing and Scanning  Out of The Box Supply Chain w
ith Testing and Scanning.
  ootb-supply-chain-testing.tanzu.vmware.com          Tanzu App Platform Out o
f The Box Supply Chain with Testing                 Out of The Box Supply Chain w
ith Testing.
  ootb-templates.tanzu.vmware.com                     Tanzu App Platform Out o
f The Box Templates                                 Out of The Box Templates.
  policy.apps.tanzu.vmware.com                        Supply Chain Security To
ols - Policy Controller                             Policy Controller enables def
ining of a policy to restrict unsigned container

images.
  scanning.apps.tanzu.vmware.com                      Supply Chain Security To
ols - Scan                                          Scan for vulnerabilities and
enforce policies directly within Kubernetes native

Supply Chains.
  service-bindings.labs.vmware.com                    Service Bindings for Kub
ernetes                                             Service Bindings for Kubernet
es implements the Service Binding Specification.
  services-toolkit.tanzu.vmware.com                   Services Toolkit
The Services Toolkit enables the management, lifecycle, discoverability and

connectivity of Service Resources (databases, message queues, DNS records,

etc.).
  snyk.scanning.apps.tanzu.vmware.com                 Snyk for Supply Chain Se
curity Tools - Scan                                 Default scan templates using
Snyk
  spring-boot-conventions.tanzu.vmware.com            Tanzu Spring Boot Conven
tions Server                                        Default Spring Boot conventio
n server.
  sso.apps.tanzu.vmware.com                           AppSSO
Application Single Sign-On for Tanzu
  tap-auth.tanzu.vmware.com                           Default roles for Tanzu
Application Platform                                Default roles for Tanzu Appli
cation Platform
  tap-gui.tanzu.vmware.com                            Tanzu Application Platfo
rm GUI                                              web app graphical user interf
ace for Tanzu Application Platform
  tap-telemetry.tanzu.vmware.com                      Telemetry Collector for
Tanzu Application Platform                          Tanzu Application Platform Te
lemetry
  tap.tanzu.vmware.com                                Tanzu Application Platfo
rm                                                  Package to install a set of T
AP components to get you started based on your use
```

```
case.
  tekton.tanzu.vmware.com                                Tekton Pipelines
Tekton Pipelines is a framework for creating CI/CD systems.
  workshops.learningcenter.tanzu.vmware.com            Workshop Building Tutori
al                                                  Workshop Building Tutorial
```

Install your Tanzu Application Platform profile

The `tap.tanzu.vmware.com` package installs predefined sets of packages based on your profile settings. This is done by using the package manager installed by Tanzu Cluster Essentials.

For more information about profiles, see Components and installation profiles.

To prepare to install a profile:

1. List version information for the package by running:

   ```
   tanzu package available list tap.tanzu.vmware.com --namespace tap-install
   ```

2. Create a `tap-values.yaml` file by using the Full Profile sample in the following section as a guide. These samples have the minimum configuration required to deploy Tanzu Application Platform. The sample values file contains the necessary defaults for:

   - The meta-package, or parent Tanzu Application Platform package.
   - Subordinate packages, or individual child packages.

   Keep the values file for future configuration use.

   Note

   `tap-values.yaml` is set as a Kubernetes secret, which provides secure means to read credentials for Tanzu Application Platform components.

3. View possible configuration settings for your package

Full profile

The following is the YAML file sample for the full-profile. The `profile:` field takes `full` as the default value, but you can also set it to `iterate`, `build`, `run` or `view`. Refer to Install multicluster Tanzu Application Platform profiles for more information.

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME"
    secret:
      name: image-registry-creds
      namespace: tap-install
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  ca_cert_data: | # To be passed if using custom certificates.
      -----BEGIN CERTIFICATE-----
      MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
      -----END CERTIFICATE-----

ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.
```

```
#The above keys are minimum numbers of entries needed in tap-values.yaml to get a func
tioning TAP Full profile installation.

#Below are the keys which may have default values set, but can be overridden.

profile: full # Can take iterate, build, run, view.

supply_chain: basic # Can take testing, testing_scanning.

ootb_supply_chain_basic: # Based on supply_chain set above, can be changed to ootb_sup
ply_chain_testing, ootb_supply_chain_testing_scanning.
  registry:
    server: "SERVER-NAME" # Takes the value from shared section above by default, but
can be overridden by setting a different value.
    repository: "REPO-NAME" # Takes the value from shared section above by default, bu
t can be overridden by setting a different value.
  gitops:
    ssh_secret: "SSH-SECRET-KEY" # Takes "" as value by default; but can be overridden
by setting a different value.

contour:
  envoy:
    service:
      type: LoadBalancer # This is set by default, but can be overridden by setting a
different value.

buildservice:
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret: # Takes the value from the shared section by default,
but can be overridden by setting a different value.
    name: image-registry-creds
    namespace: tap-install

tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml

metadata_store:
  ns_for_export_app_cert: "MY-DEV-NAMESPACE"
  app_service_type: ClusterIP # Defaults to LoadBalancer. If shared.ingress_domain is
set earlier, this must be set to ClusterIP.

scanning:
  metadataStore:
    url: "" # Configuration is moved, so set this string to empty.

grype:
  namespace: "MY-DEV-NAMESPACE"
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:
    - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.

    - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
    - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.24.x` or `1.25.x`, where `x` stands for the patch version. Examples:
    - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
    - Red Hat OpenShift Container Platform v4.12 uses the Kubernetes version `1.25.2`.
- `SERVER-NAME` is the host name of the registry server. Examples:
    - Harbor has the form `server: "my-harbor.io"`.
    - Docker Hub has the form `server: "index.docker.io"`.
    - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. If this key is passed through the shared section earlier and AWS ECR registry is used, you must ensure that the `SERVER-NAME/REPO-NAME/buildservice` and `SERVER-NAME/REPO-NAME/workloads` exist. AWS ECR expects the paths to be pre-created. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
    - Harbor has the form `repository: "my-project/supply-chain"`.
    - Docker Hub has the form `repository: "my-dockerhub-user"`.
    - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. This field is only required if you use a private repository, otherwise, leave it empty. See Git authentication for more information.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, you can use a Backstage-compliant catalog you've already built and posted on the Git infrastructure.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Store exports secrets to the namespace, and SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace. If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"` to export the SCST - Store CA certificate to all namespaces.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.

Tanzu Application Platform is part of VMware's CEIP program where data is collected to help improve the customer experience. By setting `ceip_policy_disclosed` to `true` (not a string), you acknowledge the program is disclosed to you and you are aware data collection is happening. This field must be set for the installation to be completed. See Opt out of telemetry collection for more information.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

If you use AWS, the default settings creates a classic LoadBalancer. To use the Network LoadBalancer instead of the classic LoadBalancer for ingress, add the following to your `tap-values.yaml`:

```
contour:
  infrastructure_provider: aws
  envoy:
    service:
      aws:
        LBType: nlb
```

(Optional) Additional Build Service configurations

The following tasks are optional during the Tanzu Application Platform installation process:

- (Optional) Configure your profile with full dependencies
- (Optional) Configure your profile with the Jammy stack only

(Optional) Configure your profile with full dependencies

When you install a profile that includes Tanzu Build Service, Tanzu Application Platform is installed with the `lite` set of dependencies. These dependencies consist of buildpacks and stacks required for application builds.

The `lite` set of dependencies do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

```
buildservice:
  ...
  exclude_dependencies: true
  ...
```

After configuring `full` dependencies, you must install the dependencies after you have finished installing your Tanzu Application Platform package. See Install the full dependencies package for more information.

(Optional) Configure your profile with the Jammy stack only

Tanzu Application Platform v1.5.0 supports building applications with both the Ubuntu v22.04 (Jammy) and v18.04 (Bionic) stack. For more information, see Bionic and Jammy stacks.

To install Tanzu Application Platform with Jammy as the only available stack, include the `stack_configuration: jammy-only` field under the `buildservice:` section in `tap-values.yaml`.

Security Context Constraints

Security Context Constraints (SCC) define a set of rules that a pod must satisfy to be created. Tanzu Application Platform components use the built-in nonroot-v2 or restricted-v2 SCC.

In Red Hat OpenShift, SCC are used to restrict privileges for pods. In Tanzu Application Platform v1.4 there is no custom SCC.

Tanzu Application Platform packages reconcile without any issues when using OpenShift v4.11 with restricted-v2 or nonroot-v2.

(Optional) Exclude components that require RedHat OpenShift privileged SCC

Learning Center package uses privileged SCC. To exclude this package, update your `tap-values` file with a section listing the exclusions:

```
...
excluded_packages:
  - learningcenter.tanzu.vmware.com
  - workshops.learningcenter.tanzu.vmware.com
...
```

See Exclude packages from a Tanzu Application Platform profile for more information.

Install your Tanzu Application Platform package

Follow these steps to install the Tanzu Application Platform package:

1. Install the package by running:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v $TAP_VERSION --values-file
   tap-values.yaml -n tap-install
   ```

2. Verify the package install by running:

   ```
   tanzu package installed get tap -n tap-install
   ```

   This can take 5-10 minutes because it installs several packages on your cluster.

3. Verify that the necessary packages in the profile are installed by running:

   ```
   tanzu package installed list -A
   ```

4. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

After installing the Full profile on your cluster, you can install the Tanzu Developer Tools for VS Code Extension to help you develop against it. For instructions, see Install Tanzu Developer Tools for your VS Code.

Note

You can run the following command after reconfiguring the profile to reinstall the Tanzu Application Platform:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v $TAP_VERSION  --values-f
ile tap-values.yaml -n tap-install
```

Install the full dependencies package

If you configured `full` dependencies in your `tap-values.yaml` file in Configure your profile with full dependencies earlier, you must install the `full` dependencies package.

For more information about the differences between `lite` and `full` dependencies, see About lite and full dependencies.

To install the `full` dependencies package:

1. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tap-values.yaml` file under the `buildservice` section. For example:

   ```
   buildservice:
     ...
   ```

```
  exclude_dependencies: true
  ...
```

2. Get the latest version of the `buildservice` package by running:

```
tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
all
```

3. Relocate the Tanzu Build Service full dependencies package repository by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
ps-package-repo:VERSION \
  --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tbs-full-deps
```

   Where `VERSION` is the version of the `buildservice` package you retrieved in the previous step.

4. Add the Tanzu Build Service full dependencies package repository by running:

```
tanzu package repository add tbs-full-deps-repository \
  --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tbs-full-deps:VERSION \
  --namespace tap-install
```

   Where `VERSION` is the version of the `buildservice` package you retrieved earlier.

5. Install the full dependencies package by running:

```
tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
N -n tap-install
```

   Where `VERSION` is the version of the `buildservice` package you retrieved earlier.

Access Tanzu Application Platform GUI

To access Tanzu Application Platform GUI, you can use the host name that you configured earlier. This host name is pointed at the shared ingress. To configure LoadBalancer for Tanzu Application Platform GUI, see Access Tanzu Application Platform GUI.

You're now ready to start using Tanzu Application Platform GUI. Proceed to the Getting Started topic or the Tanzu Application Platform GUI - Catalog Operations topic.

Exclude packages from a Tanzu Application Platform profile

To exclude packages from a Tanzu Application Platform profile:

1. Find the full subordinate (child) package name:

```
tanzu package available list --namespace tap-install
```

2. Update your `tap-values` file with a section listing the exclusions:

```
profile: PROFILE-VALUE
excluded_packages:
  - tap-gui.tanzu.vmware.com
  - service-bindings.lab.vmware.com
```

Important

If you exclude a package after performing a profile installation including that package, you cannot see the accurate package states immediately after running `tap package installed list -n tap-install`. Also, you can break package dependencies by removing a package. Allow 20 minutes to verify that all packages have reconciled correctly while troubleshooting.

View possible configuration settings for your package

To view possible configuration settings for a package, run:

```
tanzu package available get tap.tanzu.vmware.com/$TAP_VERSION --values-schema --namesp
ace tap-install
```

Note

The `tap.tanzu.vmware.com` package does not show all configuration settings for packages it plans to install. The package only shows top-level keys. You can view individual package configuration settings with the same `tanzu package available get` command. For example, to find the keys for Cloud Native Runtimes, you must first identify the version of the package with `tanzu package installed list -n tap-install`, which lists all the installed packages versions. Then run the command `tanzu package available get -n tap-install cnrs.tanzu.vmware.com/CNRS-VERSION --values-schema` by using the package version listed for Cloud Native Runtimes.

```
profile: full

# Shared configurations go under the shared key.
shared:
  ingress_domain: tap.example.com

# ...

# For example, CNRs specific values go under its name.
cnrs:
  provider: local

# For example, App Accelerator specific values go under its name.
accelerator:
  server:
    service_type: "ClusterIP"
```

Shared Keys define values that configure multiple packages. These keys are defined under the `shared` Top-level Key, as summarized in the following table:

Shared Key	Description	Optional
`ca_cert_data`	PEM-encoded certificate data to trust TLS connections with a private CA. This shared key is used by `convention_controller`, `scanning` and `source_controller`	Yes
`ingress_domain`	Domain name to be used in service routes and host names for instances of Tanzu Application Platform components.	Yes
`ingress_issuer`	A `cert-manager.io/v1/ClusterIssuer` for issuing TLS certificates to Tanzu Application Platform components. Default value: `tap-ingress-selfsigned`	Yes

Shared Key	Description	Optional
`kubernetes_distribution`	Type of Kubernetes infrastructure being used. You can use this shared key in coordination with the `kubernetes_version` key. Supported value: `openshift`.	Yes
`kubernetes_version`	Kubernetes version. You can use this shared key independently or in coordination with the `kubernetes_distribution` key. Supported value: `1.24.x`, where `x` stands for the Kubernetes patch version.	Yes
`image_registry.project_path`	Project path in the container image registry server used for builder and application images.	Yes
`image_registry.username`	User name for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`image_registry.password`	Password for the container image registry. Mutually exclusive with `shared.image_registry.secret.name/namespace`	Yes
`secret.name`	Secret name for the container image registry credentials of type `kubernetes.io/dockerconfigjson`. Mutually exclusive with `shared.image_registry.username/password`	Yes
`secret.namespace`	Secret namespace for the container image registry credentials. Mutually exclusive with `shared.image_registry.username/password`	Yes
`activateAppLiveViewSecureAccessControl`	Enable secure access connection between Application Live View components.	Yes

The following table summarizes the top-level keys used for package-specific configuration within your `tap-values.yaml`.

Package	Top-level Key
See table above.	`shared`
API Auto Registration	`api_auto_registration`
API portal	`api_portal`
Application Accelerator	`accelerator`
Application Live View	`appliveview`
Application Live View connector	`appliveview_connector`
Application Live View conventions	`appliveview-conventions`
Cartographer	`cartographer`
Cloud Native Runtimes	`cnrs`
Source Controller	`source_controller`
Supply Chain	`supply_chain`
Supply Chain Basic	`ootb_supply_chain_basic`
Supply Chain Testing	`ootb_supply_chain_testing`
Supply Chain Testing Scanning	`ootb_supply_chain_testing_scanning`
Supply Chain Security Tools - Scan	`scanning`
Supply Chain Security Tools - Scan (Grype Scanner)	`grype`
Supply Chain Security Tools - Store	`metadata_store`
Build Service	`buildservice`

Package	Top-level Key
Tanzu Application Platform GUI	`tap_gui`
Learning Center	`learningcenter`

For information about package-specific configuration, see Install individual packages.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service
- Install Application Live View
- Install Application Single Sign-On
- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform
- Install Developer Conventions
- Install Eventing
- Install Flux CD Source Controller
- Install Learning Center for Tanzu Application Platform
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit
- Install Source Controller
- Install Spring Boot conventions

- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Application Platform GUI
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package installed list --namespace tap-install
   \ Retrieving installed packages...
   NAME                     PACKAGE-NAME                                      PAC
   KAGE-VERSION  STATUS
   api-portal               api-portal.tanzu.vmware.com                       1.
   0.3           Reconcile succeeded
   app-accelerator          accelerator.apps.tanzu.vmware.com                 1.
   0.0           Reconcile succeeded
   app-live-view            appliveview.tanzu.vmware.com                      1.
   0.2           Reconcile succeeded
   appliveview-conventions  build.appliveview.tanzu.vmware.com                1.
   0.2           Reconcile succeeded
   cartographer             cartographer.tanzu.vmware.com                     0.
   1.0           Reconcile succeeded
   cloud-native-runtimes    cnrs.tanzu.vmware.com                             1.
   0.3           Reconcile succeeded
   convention-controller    controller.conventions.apps.tanzu.vmware.com      0.
   7.0           Reconcile succeeded
   developer-conventions    developer-conventions.tanzu.vmware.com            0.
   3.0-build.1   Reconcile succeeded
   grype-scanner            grype.scanning.apps.tanzu.vmware.com              1.
   0.0           Reconcile succeeded
   image-policy-webhook     image-policy-webhook.signing.apps.tanzu.vmware.com 1.
   1.2           Reconcile succeeded
   metadata-store           metadata-store.apps.tanzu.vmware.com              1.
   0.2           Reconcile succeeded
   ootb-supply-chain-basic  ootb-supply-chain-basic.tanzu.vmware.com          0.
   5.1           Reconcile succeeded
   ootb-templates           ootb-templates.tanzu.vmware.com                   0.
   5.1           Reconcile succeeded
   scan-controller          scanning.apps.tanzu.vmware.com                    1.
   0.0           Reconcile succeeded
   service-bindings         service-bindings.labs.vmware.com                  0.
   5.0           Reconcile succeeded
   services-toolkit         services-toolkit.tanzu.vmware.com                 0.
   8.0           Reconcile succeeded
   source-controller        controller.source.apps.tanzu.vmware.com           0.
   2.0           Reconcile succeeded
   sso4k8s-install          sso.apps.tanzu.vmware.com                         1.
   0.0-beta.2-31  Reconcile succeeded
   ```

```
tap-gui                   tap-gui.tanzu.vmware.com                              0.
3.0-rc.4       Reconcile succeeded
tekton-pipelines          tekton.tanzu.vmware.com                               0.3
0.0            Reconcile succeeded
tbs                       buildservice.tanzu.vmware.com                         1.
5.0             Reconcile succeeded
```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

For details about how to automatically set up your developer namespaces, see Provision developer namespaces in Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see Legacy namespace setup.

Next steps

- Install Tanzu Developer Tools for your VS Code

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

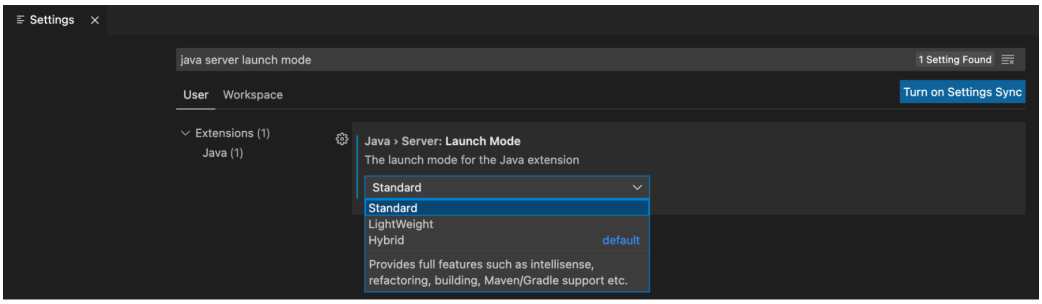
Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

Install

To install the extension:

1. Sign in to VMware Tanzu Network and download Tanzu Developer Tools for Visual Studio Code.
2. Open VS Code.
3. Press cmd+shift+P to open the Command Palette and run `Extensions: Install from VSIX...`.

4. Select the extension file **tanzu-vscode-extension.vsix**.
5. If you do not have the following extensions, and they do not automatically install, install them from VS Code Marketplace:
    - Debugger for Java
    - Language Support for Java(™) by Red Hat
    - YAML
6. Ensure Language Support for Java is running in Standard Mode. You can configure it in the **Settings** menu by going to **Code** > **Preferences** > **Settings** under **Java > Server: Launch Mode**.



When the JDK and Language Support for Java are configured correctly, you see that the integrated development environment creates a directory target where the code is compiled.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: For more information, see Integrating Live Hover by using Spring Boot Tools. Reload VS Code for this change to take effect.
    - **Source Image**: (Required) The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name.

- **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
- **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in `kubeconfig` is the default.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and select **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Custom Security Context Constraint details for Tanzu Application Platform

Custom Security Context Constraint (commonly known as SCC) details for Tanzu Application Platform (commonly known as TAP) components are as follows:

- Application Accelerator on OpenShift cluster
- Application Live View on OpenShift
- Application Single Sign-On for OpenShift cluster
- Contour for OpenShift cluster
- Developer Conventions for OpenShift cluster
- Tanzu Build Service for OpenShift cluster

Application Accelerator on OpenShift

On OpenShift clusters, Application Accelerator must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes pod security standards. Tanzu Application Platform configures the following SCC for Application Accelerator when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
#@ load("@ytt:data", "data")
#@ load("@ytt:assert", "assert")

#@ kubernetes_distribution = data.values.kubernetes_distribution
#@ validDistributions = [None, "", "openshift"]
#@ if kubernetes_distribution not in validDistributions:
#@   assert.fail("{} not in {}".format(kubernetes_distribution, validDistributions))
#@ end

#@ if kubernetes_distribution == "openshift":
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: accelerator-system-nonroot-scc
  namespace: accelerator-system
rules:
```

```
- apiGroups:
  - security.openshift.io
  resourceNames:
  - nonroot
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: accelerator-system-nonroot-scc
  namespace: accelerator-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: accelerator-system-nonroot-scc
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts:accelerator-system
#@ end
```

Application Live View on OpenShift

Application Live View must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards on Openshift. Tanzu Application Platform configures the following SCC for Application Live View back end, Application Live View connector, and Application Live View convention service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

The following is a `SecurityContextConstraints` specification for Application Live View connector:

```
---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: appliveview-connector-restricted-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - configMap
  - downwardAPI
  - emptyDir
```

```
  - persistentVolumeClaim
  - projected
  - secret
seccompProfiles:
  - runtime/default
```

The preceding `SecurityContextConstraints` specification is applicable to Application Live View back end and Application Live View convention service as well.

Application Single Sign-On for OpenShift cluster

On OpenShift clusters, AppSSO must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards. Tanzu Application Platform configures the following SCC for AppSSO controller and its `AuthServer` managed resources when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: appsso-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - KILL
  - MKNOD
  - SETUID
  - SETGID
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
seccompProfiles:
  - 'runtime/default'
```

AppSSO controller's `ServiceAccount` is given the following additional permissions, including a `use` permission for AppSSO SCC, so `AuthServer` can use the custom SCC:

```
- apiGroups:
    - security.openshift.io
  resources:
    - securitycontextconstraints
```

```
  verbs:
    - "get"
    - "list"
    - "watch"
```

```
- apiGroups:
    - security.openshift.io
  resourceNames:
    - appsso-scc
  resources:
    - securitycontextconstraints
  verbs:
    - "use"
```

Contour for OpenShift cluster

On OpenShift clusters, Contour must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards. Tanzu Application Platform configures the following SCC for the service accounts in the `tanzu-system-ingress` namespace, which applies to Contour's controller and Envoy pods, when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    kubernetes.io/description: nonroot provides all features of the restricted SCC
      but allows users to run with any non-root UID.  The user must specify the UID
      or it must be specified on the by the manifest of the container runtime. On
      top of the legacy 'nonroot' SCC, it also requires to drop ALL capabilities and
      does not allow privilege escalation binaries. It will also default the seccomp
      profile to runtime/default if unset, otherwise this seccomp profile is required.
  name: contour-seccomp-nonroot-v2
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
- NET_BIND_SERVICE
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups: []
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
- ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
seccompProfiles:
- runtime/default
supplementalGroups:
  type: RunAsAny
```

```
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

The SCC is bound to the service accounts by using the following Role and RoleBinding:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: contour-seccomp-nonroot-v2
  namespace: tanzu-system-ingress
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - contour-seccomp-nonroot-v2
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: contour-seccomp-nonroot-v2
  namespace: tanzu-system-ingress
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: contour-seccomp-nonroot-v2
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts:tanzu-system-ingress
```

Developer Conventions for OpenShift cluster

On OpenShift clusters, Developer Conventions must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes pod security standards. Tanzu Application Platform configures the following SCC for the Developer Convention's webhook when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: developer-conventions-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
```

```
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - secret
seccompProfiles: []
groups:
  - system:serviceaccounts:developer-conventions
```

Tanzu Build Service for OpenShift cluster

On OpenShift clusters Tanzu Build Service must run with a custom Security Context Constraint (SCC) to enable compliance. Tanzu Application Platform configures the following SCC for Tanzu Build Service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

```
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: tbs-restricted-scc-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
  - NET_BIND_SERVICE
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups: []
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
seccompProfiles:
  - runtime/default
supplementalGroups:
  type: RunAsAny
users: []
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
```

It also applies the following RBAC to allow Tanzu Build Service services to use the SCC:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    apps.tanzu.vmware.com/aggregate-to-workload: "true"
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  name: system:tbs:scc:restricted-with-seccomp
rules:
  - apiGroups:
      - security.openshift.io
    resourceNames:
      - tbs-restricted-scc-with-seccomp
    resources:
      - securitycontextconstraints
    verbs:
      - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:tbs:scc:restricted-with-seccomp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:tbs:scc:restricted-with-seccomp
subjects:
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-controller-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: secret-syncer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: warmer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: build-service-daemonset-serviceaccount
  - kind: ServiceAccount
    namespace: cert-injection-webhook
    name: cert-injection-webhook-sa
  - kind: ServiceAccount
    namespace: kpack
    name: kp-default-repository-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: kpack-pull-lifecycle-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: controller
  - kind: ServiceAccount
    namespace: kpack
    name: webhook
  - kind: ServiceAccount
    namespace: stacks-operator-system
    name: controller-manager
```

Install Tanzu Application Platform (GitOps)

GitOps is a set of practices and principles to manage Kubernetes infrastructure and application deployments using Git as the single source of truth. It promotes declarative configurations and automated workflows to ensure consistency, reliability, and traceability for your application deployments.

The key components involved in implementing GitOps with Kubernetes include:

- **Git as the single source of truth**: The desired state is stored in a Git repository. To change the cluster state, you must change it in the Git repository instead of modifying it directly on the cluster.
- **Declarative configuration**: GitOps follows a declarative approach, where the desired state is defined in the declarative configuration files.
- **Pull-based synchronization**: GitOps follows a pull-based model. Kubernetes cluster periodically pulls the desired state from the Git repository. This approach ensures that the cluster is always in sync with the desired configuration.

Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

How Tanzu RI supports GitOps

The Tanzu GitOps Reference Implementation (RI) is built upon Carvel, which shares the same packaging APIs as the Tanzu Application Platform. Carvel packaging APIs support all the GitOps features and enables a native GitOps flow.

- All the packaging APIs are declarative in nature.
- Among many options to fetch the manifest to be deployed, it can also pull the content from the Git repository, making Git the source of truth.
- Packages installed are reconciled every time after the SyncPeriod expires (10 minutes by default). As part of the reconciliation, it fetches the manifest from the Git repository and when the desired state is different from the actual state on Kubernetes, it converges the resources to their desired state declared in Git.

GitOps benefits

GitOps offers the following benefits:

- **Compliance and auditing capabilities**: In GitOps, Git is the single source of truth, enabling auditors to access a complete audit trail of all configuration changes.
- **Disaster recovery**: Disaster recovery involves an organization's efforts to restore access and function to its IT infrastructure. With all configurations securely stored in Git, disaster recovery becomes as straightforward as reapplying the desired configuration version.
- **Repeatable**: Running Tanzu CLI commands with environment variables or configuration files on a local machine is no longer required. Instead, all the necessary configurations and service accounts for access are configured in a shared Git repository. This approach allows any operator to make edits to a file, and the system's behavior remains independent of their local environment.

GitOps install paths

Choose one of the following install paths to install Tanzu Application Platform on your Kubernetes clusters through GitOps:

GitOps with Secrets OPerationS (SOPS)

Applies to the scenario when you want a simple instance and store sensitive data encrypted in your Git repo:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Install Tanzu Application Platform.	Install Tanzu Application Platform through Gitops with Secrets OPerationS (SOPS)
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

GitOps with External Secrets Operator (ESO)

Applies to the scenario when you want to store sensitive data in external store:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create AWS Resources (EKS Cluster, roles, etc)	Create AWS Resources
4.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install Tanzu Application Platform through GitOps with External Secrets Operator (ESO)
6.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
7.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

* *When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform (GitOps)

GitOps is a set of practices and principles to manage Kubernetes infrastructure and application deployments using Git as the single source of truth. It promotes declarative configurations and automated workflows to ensure consistency, reliability, and traceability for your application deployments.

The key components involved in implementing GitOps with Kubernetes include:

- **Git as the single source of truth**: The desired state is stored in a Git repository. To change the cluster state, you must change it in the Git repository instead of modifying it directly on the cluster.
- **Declarative configuration**: GitOps follows a declarative approach, where the desired state is defined in the declarative configuration files.
- **Pull-based synchronization**: GitOps follows a pull-based model. Kubernetes cluster periodically pulls the desired state from the Git repository. This approach ensures that the cluster is always in sync with the desired configuration.

Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

How Tanzu RI supports GitOps

The Tanzu GitOps Reference Implementation (RI) is built upon Carvel, which shares the same packaging APIs as the Tanzu Application Platform. Carvel packaging APIs support all the GitOps features and enables a native GitOps flow.

- All the packaging APIs are declarative in nature.
- Among many options to fetch the manifest to be deployed, it can also pull the content from the Git repository, making Git the source of truth.
- Packages installed are reconciled every time after the SyncPeriod expires (10 minutes by default). As part of the reconciliation, it fetches the manifest from the Git repository and when the desired state is different from the actual state on Kubernetes, it converges the resources to their desired state declared in Git.

GitOps benefits

GitOps offers the following benefits:

- **Compliance and auditing capabilities**: In GitOps, Git is the single source of truth, enabling auditors to access a complete audit trail of all configuration changes.
- **Disaster recovery**: Disaster recovery involves an organization's efforts to restore access and function to its IT infrastructure. With all configurations securely stored in Git, disaster recovery becomes as straightforward as reapplying the desired configuration version.
- **Repeatable**: Running Tanzu CLI commands with environment variables or configuration files on a local machine is no longer required. Instead, all the necessary configurations and service accounts for access are configured in a shared Git repository. This approach allows any operator to make edits to a file, and the system's behavior remains independent of their local environment.

GitOps install paths

Choose one of the following install paths to install Tanzu Application Platform on your Kubernetes clusters through GitOps:

GitOps with Secrets OPerationS (SOPS)

Applies to the scenario when you want a simple instance and store sensitive data encrypted in your Git repo:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
4.	Install Tanzu Application Platform.	Install Tanzu Application Platform through Gitops with Secrets OPerationS (SOPS)
5.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
6.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
7.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

GitOps with External Secrets Operator (ESO)

Applies to the scenario when you want to store sensitive data in external store:

Step	Task	Link
1.	Review the prerequisites to ensure you have met all requirements before installing.	Prerequisites
2.	Accept Tanzu Application Platform EULAs and install the Tanzu CLI.	Accept Tanzu Application Platform EULAs and installing the Tanzu CLI
3.	Create AWS Resources (EKS Cluster, roles, etc)	Create AWS Resources
4.	Install Cluster Essentials for Tanzu*.	Deploy Cluster Essentials
5.	Add the Tanzu Application Platform package repository, prepare your Tanzu Application Platform profile, and install the profile to the cluster.	Install Tanzu Application Platform through GitOps with External Secrets Operator (ESO)
6.	(Optional) Install any additional packages that were not in the profile.	Install individual packages
7.	Set up developer namespaces to use your installed packages.	Set up developer namespaces to use your installed packages
8.	Install developer tools into your integrated development environment (IDE).	Install Tanzu Developer Tools for your VS Code

* *When you use a VMware Tanzu Kubernetes Grid cluster, there is no need to install Cluster Essentials because the contents of Cluster Essentials are already installed on your cluster.*

After installing Tanzu Application Platform on to your Kubernetes clusters, proceed with Get started with Tanzu Application Platform.

Install Tanzu Application Platform through GitOps with External Secrets Operator (ESO)

Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) through GitOps with secrets managed in an external secrets store. To decide which approach to use, see Choosing SOPS or ESO.

Tanzu GitOps Reference Implememtation (RI) does not support changing the secrets management strategy for a cluster. The External Secrets Operator integration in this release of Tanzu GitOps RI is verified to support AWS Elastic Kubernetes Service cluster with AWS Secrets Manager. Other combinations of Kubernetes distribution and ESO providers are not verified.

Prerequisites

Before installing Tanzu Application Platform, ensure you have:

- Completed the Prerequisites.
- Created AWS Resources.
- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.
- Installed Cluster Essentials for Tanzu.
- Installed eksctl CLI.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following documentation for a registry to learn how to set it up:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

```
export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
```

```
export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
export TAP_VERSION=VERSION-NUMBER
export INSTALL_REPO=TARGET-REPOSITORY
```

Where:

- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is your own container registry.
- `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`.
- `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
- `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.
- `TARGET-REPOSITORY` is your target repository, a folder or repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

VMware recommends using a JSON key file to authenticate with Google Container Registry. In this case, the value of `INSTALL_REGISTRY_USERNAME` is `_json_key` and the value of `INSTALL_REGISTRY_PASSWORD` is the content of the JSON key file. For more information about how to generate the JSON key file, see Google Container Registry documentation.

2. Install the Carvel tool `imgpkg` CLI.

   To query for the available versions of Tanzu Application Platform on VMWare Tanzu Network Registry, run:

   ```
   imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/tap-pac
   kages | sort -V
   ```

3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-package
   s:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-pac
   kages
   ```

(Optional) Install Tanzu Application Platform in an air-gapped environment

Complete the following steps if you install Tanzu Application Platform in an air-gapped environment:

1. Relocate the Tanzu Build Service images to your registry:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
   ps-package-repo:VERSION --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/
   full-tbs-deps-package-repo
   ```

   Where:

   - `VERSION` is the version of Tanzu Build Service. You can retrieve this value by running `kubectl get package -n tap-install | grep buildservice`

2. Configure custom certificate authorities for Tanzu Application Platform GUI.
3. Host a `grype` database in the air-gapped environment. For more information, see Use Grype in offline and air-gapped environments.

Create a new Git repository

1. In a hosted Git service, for example, GitHub or GitLab, create a new respository.

   This version of Tanzu GitOps RI only supports authenticating to a hosted Git repository by using SSH.

2. Initialize a new Git repository:

   ```
   mkdir -p $HOME/tap-gitops
   cd $HOME/tap-gitops

   git init
   git remote add origin git@github.com:my-organization/tap-gitops.git
   ```

3. Create a read-only deploy key for this new repository (recommended) or SSH key for an account with read access to this repository.

   The private portion of this key is referred to as `GIT_SSH_PRIVATE_KEY`.

Download and unpack Tanzu GitOps Reference Implementation (RI)

1. Sign in to VMware Tanzu Network.
2. Go to the Tanzu Application Platform product page.
3. Select **Release 1.5.12** from the release drop-down menu.
4. Click **Tanzu GitOps Reference Implementation**.
5. Unpack the downloaded TGZ file into the `$HOME/tap-gitops` directory by running:

   ```
   tar -xvf tanzu-gitops-ri-*.tgz -C $HOME/tap-gitops
   ```

6. Commit the initial state:

   ```
   cd $HOME/tap-gitops

   git add . && git commit -m "Initialize Tanzu GitOps RI"
   git push -u origin
   ```

Create cluster configuration

1. Seed configuration for a cluster using ESO through the provided convenience script:

   ```
   cd $HOME/tap-gitops

   ./setup-repo.sh CLUSTER-NAME eso
   ```

   Where:

   - `CLUSTER-NAME` is the name for your cluster. Typically, this is the same as your EKS cluster's name, the name of the cluster as it appears in `eksctl get clusters`.
   - `eso` selects the External Secrets Operator-based secrets management variant.

   For example, if the name of your cluster is `iterate-green`:

   ```
   cd $HOME/tap-gitops
   ```

```
./setup-repo.sh iterate-green eso
```

This script creates the directory `clusters/iterate-green/` and copies in the configuration required to sync this Git repository with the cluster and installing Tanzu Application Platform.

2. Commit and push:

```
git add . && git commit -m 'Add "iterate-green" cluster'
git push
```

Saving the base configuration in an initial commit makes it easier to review customizations in the future.

Customize cluster configuration

Configuring the Tanzu Application Platform installation involves setting up two components:

- an installation of Tanzu Application Platform;
- an instance of Tanzu Sync, the component that implements the GitOps workflow, fetching configuration from Git and applying it to the cluster.

Follow these steps to customize your Tanzu Application Platform cluster configuration:

1. Navigate to the created directory:

```
cd clusters/CLUSTER-NAME
```

For example, if the name of your cluster is `iterate-green`:

```
cd clusters/iterate-green
```

2. Define the following environment variables:

```
export AWS_ACCOUNT_ID=MY-AWS-ACCOUNT-ID
export AWS_REGION=AWS-REGION
export EKS_CLUSTER_NAME=EKS-CLUSTER-NAME
export TAP_PKGR_REPO=TAP-PACKAGE-OCI-REPOSITORY
```

Where:

- `MY-AWS-ACCOUNT-ID` is your AWS account ID as it appears in the output of `aws sts get-caller-identity`.
- `AWS-REGION` is the region where the Secrets Manager is and the EKS cluster was created.
- `EKS-CLUSTER-NAME` is the name of the target cluster as it appears in the output of `eksctl get clusters`.
- `TAP-PACKAGE-OCI-REPOSITORY` is the fully-qualified path to the OCI repository hosting the Tanzu Application Platform images. If they are relocated to a different registry as described in Relocate images to a registry, the value is `${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages`.

Grant read access to secret data

All sensitive configuration is stored in AWS Secrets Manager secrets. Both Tanzu Sync and the Tanzu Application Platform installation require access to this sensitive data.

Follow these step to configure the IAM Role for a Service Account:

1. In AWS Identity and Access Manager, create two IAM Policies, one to read Tanzu Sync secrets and another to read the Tanzu Application Platform installation secrets by using the supplied script:

   ```
   tanzu-sync/scripts/aws/create-policies.sh
   ```

2. Create two IAM Role-to-Service Account pairs for your cluster, one for Tanzu Sync and anther for the Tanzu Application Platform installation by using the supplied script:

   ```
   tanzu-sync/scripts/aws/create-irsa.sh
   ```

   For example, if the name of the EKS cluster is `iterate-green` using the defaults, there are two IAM roles in the AWS account:

   ```
   $ aws iam list-roles --query 'Roles[?starts_with(RoleName,`iterate-green`)]'
   [
       {
           "RoleName": "iterate-green--tanzu-sync-secrets",
           ...
       },
       {
           "RoleName": "iterate-green--tap-install-secrets",
           ...
       }
   ]
   ```

Generate default configuration

You can use the following script to generate default configuration for the both Tanzu Sync and Tanzu Application Platform installation:

```
tanzu-sync/scripts/configure.sh
```

The following sections guide you through the process of editing the configuration values to suit your specific needs.

Review and store Tanzu Sync config

Configuration for Tanzu is stored in two locations:

- sensitive configuration is stored in AWS Secrets Manager;
- non-sensitive configuration are stored in YAML files in the Git repository.

Follow these steps to create the sensitive configuration and review the non-sensitive configuration:

1. Save the credentials that Tanzu Sync uses to authenticate with the Git repository:

   Create a secret named `dev/EKS-CLUSTER-NAME/tanzu-sync/sync-git-ssh` containing the following information as plaintext:

   ```
   {
    "ssh-privatekey": "... (private key portion here) ...",
    "ssh-knownhosts": "... (known_hosts for git host here) ..."
   }
   ```

   Where `EKS-CLUSTER-NAME` is the name as it appears in `eksctl get clusters`.

   For example, if the Git repository is hosted on GitHub, and the private key created in Create a new Git repository is stored in the file `~/.ssh/id_ed25519`:

```
aws secretsmanager create-secret \
   --name dev/${EKS_CLUSTER_NAME}/tanzu-sync/sync-git-ssh \
   --secret-string "$(cat <<EOF
{
 "ssh-privatekey": "$(cat ~/.ssh/id_ed25519 | awk '{printf "%s\\n", $0}')",
 "ssh-knownhosts": "$(ssh-keyscan github.com | awk '{printf "%s\\n", $0}')"
}
EOF
)"
```

Where:

- The content of `~/.ssh/id_ed25519` is the private portion of the SSH key.
- `ssh-keyscan` obtains the public keys for the SSH host.
- `awk '{printf "%s\n", $0}'` converts a multiline string into a single-line string with embedded newline chars (`\n`). JSON does not support multiline strings.

Caution

This version of Tanzu GitOps RI only supports authenticating to a hosted Git repository by using SSH. Authenticating by using HTTP Basic Authentication is not supported.

2. Save the authentication credentials required for accessing the OCI registry that hosts the Tanzu Application Platform images by creating a secret named `dev/EKS-CLUSTER-NAME/tanzu-sync/install-registry-dockerconfig` containing the following information as plaintext:

```
{
  "auths": {
    "MY-REGISTRY": {
      "username": "MY-REGISTRY-USER",
      "password": "MY-REGISTRY-PASSWORD"
    }
  }
}
```

Where:

- `EKS-CLUSTER-NAME` is the name as it appears in `eksctl get clusters`
- `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is the container registry to which the Tanzu Application Platform images are located.

For example:

```
aws secretsmanager create-secret \
   --name dev/${EKS_CLUSTER_NAME}/tanzu-sync/install-registry-dockerconfig \
   --secret-string "$(cat <<EOF
{
 "auths": {
   "${INSTALL_REGISTRY_HOSTNAME}": {
     "username": "${INSTALL_REGISTRY_USERNAME}",
     "password": "${INSTALL_REGISTRY_PASSWORD}"
   }
 }
}
```

```
EOF
)"
```

3. Review the hosted Git URL and branch Tanzu Sync should use.

   This configuration was generated by the `configure.sh` script. It reported:

   ```
   ...
   wrote non-sensitive Tanzu Sync configuration to: tanzu-sync/app/values/tanzu-sy
   nc.yaml
   ...
   ```

   For example, for the `iterate-green` cluster, if the Git repository is hosted on GitHub at `my-organization/tap-gitops` on the `main` branch, `tanzu-sync.yaml` contains the following information:

   ```
   ---
   git:
    url: git@github.com:my-organization/tap-gitops.git
    ref: origin/main
    sub_path: clusters/iterate-green/cluster-config
   ```

   You can review and edit these values as needed.

4. Review the integration with External Secrets Operator.

   This configuration was generated by the `configure.sh` script. It reported:

   ```
   ...
   wrote ESO configuration for Tanzu Sync to: tanzu-sync/app/values/tanzu-sync-es
   o.yaml
   ...
   ```

   For example, for the `iterate-green` cluster, if the AWS account is `665100000000`, `tanzu-sync-eso.yaml` contains the following information:

   ```
   ---
   secrets:
    eso:
      aws:
        region: us-west-2
        tanzu_sync_secrets:
          role_arn: arn:aws:iam::665100000000:role/iterate-green--tanzu-sync-secre
   ts
      remote_refs:
        sync_git_ssh:
          ssh_private_key:
            key: dev/iterate-green/tanzu-sync/sync-git-ssh
            property: ssh-privatekey
          ssh_known_hosts:
            key: dev/iterate-green/tanzu-sync/sync-git-ssh
            property: ssh-knownhosts
        install_registry_dockerconfig:
          dockerconfigjson:
            key: dev/iterate-green/tanzu-sync/install-registry-dockerconfig
   ```

   Where:

   - `role_arn` is the IAM role that grants permission to Tanzu Sync to read secrets specific to Tanzu Sync. This role was created in the Grant read access to secret data section.

- `ssh_private_key` is the AWS Secrets Manager secret name, as known as key, and JSON property that contains the private key portion of the SSH authentication to the Git repository created earlier.
- `ssh_known_hosts` is the AWS Secrets Manager secret name, as known as key, and JSON property that contains the known host entries for the SSH authentication to the Git repository created earlier.
- `install_registry_dockerconfig` contains the AWS Secrets Manager secret name that contains the Docker config authentication to the OCI registry hosting the Tanzu Application Platform images created earlier.

5. Commit the Tanzu Sync configuration.

   For example, for the “iterate-green” cluster, run:

   ```
   git add tanzu-sync/
   git commit -m 'Configure Tanzu Sync on "iterate-green"'
   ```

Review and store Tanzu Application Platform installation config

Configuration for the Tanzu Application Platform installation are stored in two places:

- sensitive configuration is stored in AWS Secrets Manager;
- non-sensitive configuration is stored in YAML files in the Git repository.

Follow these steps to create the sensitive configuration and review the non-sensitive configuration:

1. Create a secret named `dev/${EKS_CLUSTER_NAME}/tap/sensitive-values.yaml` that stores the sensitive data such as username, password, private key from the `tap-values.yaml` file:

   ```
   aws secretsmanager create-secret \
      --name dev/${EKS_CLUSTER_NAME}/tap/sensitive-values.yaml \
      --secret-string "$(cat <<EOF
   ---
   # this document is intentionally initially blank.
   EOF
   )"
   ```

   You can start with an empty document and edit it later on in the Configure and push the Tanzu Application Platform values section.

2. Review the integration with External Secrets Operator.

   This configuration was generated by the `configure.sh` script. It reported:

   ```
   ...
   wrote ESO configuration for TAP Install to: cluster-config/values/tap-install-e
   so-values.yaml
   ...
   ```

   For example, for the `iterate-green` cluster, if the AWS account is `665100000000`, `tap-install-eso-values.yaml` contains the following information:

   ```
   ---
   tap_install:
    secrets:
      eso:
        aws:
          region: us-west-2
          tap_install_secrets:
            role_arn: arn:aws:iam:665100000000:iterate-green--tap-install-secrets
        remote_refs:
   ```

```
    tap_sensitive_values:
      sensitive_tap_values_yaml:
        key: dev/iterate-green/tap/sensitive-values.yaml
```

Where:

- `role_arn` is the IAM role that grants permission to Tanzu Application Platform installation to read its associated secrets. This role was created in the Grant read access to secret data section.
- `sensitive_tap_values_yaml.key` is the AWS Secrets Manager secret name that contains the sensitive data from the `tap-values.yaml` file for this cluster in a YAML format.

3. Commit the Tanzu Application Platform installation configuration.

   For example, for the `iterate-green` cluster, run:

   ```
   git add cluster-config/
   git commit -m 'Configure installer for TAP 1.5.0 on "iterate-green"'
   ```

Configure and push the Tanzu Application Platform values

The configuration for the Tanzu Application Platform is divided into two separate locations:

- sensitive configuration is stored in a AWS Secrets Manager secret created as described in the Review and store Tanzu Application Platform installation config section.
- non-sensitive configuration is stored in a plain YAML file `cluster-config/values/tap-values.yaml`

Follow these steps to split the Tanzu Application Platform values:

1. Create the file `cluster-config/values/tap-values.yaml` by using the Full Profile (AWS) which contains the minimum configurations required to deploy Tanzu Application Platform on AWS.

   The Tanzu Application Platform values are input configurations to the Tanzu Application Platform installation and are placed under the `tap_install.values` path.

   ```
   tap_install:
     values:
       # Tanzu Application Platform values go here.
       shared:
         ingress_domain: "INGRESS-DOMAIN"
       ceip_policy_disclosed: true
   ...
   ```

   To install Tanzu Application Service in an offline environment, you must configure `Tanzu Build Service` and `Grype` to work in an air-gapped environment:

   ```
   ---
   tap_install:
     values:
       ...
       buildservice:
         exclude_dependencies: true
       grype:
         db:
           dbUpdateUrl: INTERNAL-VULN-DB-URL
   ```

   Where:

- `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

For more information, see Components and installation profiles.

2. (Optional) Update Tanzu Application Platform to use the latest patch:

```
tap_install:
    ...
    version:
        package_repo_bundle_tag: "1.5.6" # Populate these values with the lates
t patch version.
        package_version: "1.5.6"
```

Where:

- `package_repo_bundle_tag` is the version of Tanzu Application Platform you want to upgrade to.
- `package_version` is the version of Tanzu Application Platform you want to upgrade to. This version must match `package_repo_bundle_tag`.

Note

Tanzu GitOps RI does not provide a separate artifact for each patch version within a minor line. For example, Tanzu Application Platform v1.5.x contains the GitOps artifact with v1.5.0 only.

3. Review the contents of `tap-values.yaml` and move all sensitive values into the AWS Secrets Store secret created in the Review and store Tanzu Application Platform installation config section.

   For example, if the `iterate-green` cluster is configured with the basic Out of the Box Supply Chain, this might include a passphrase for that supply chain's GitOps flow:

```
---
tap_install:
 values:
   ...
   ootb_supply_chain_basic:
     registry:
       server: "SERVER-NAME"
       repository: "REPO-NAME"
     gitops:
       ssh_secret: "SSH-SECRET-KEY"   # <== sensitive value; do not commit to G
it repository!
   ...
```

   To maintain the secrecy of `ootb_supply_chain_basic.gitops.ssh_secret`, move this value from the `tap-values.yaml` file:

```
---
tap_install:
 values:
   ...
   ootb_supply_chain_basic:
     registry:
       server: "SERVER-NAME"
       repository: "REPO-NAME"
   ...
```

Add it to the AWS Secrets Store secret named `dev/iterate-green/tap/sensitive-values.yaml`, by default, without the `tap_install.values` root:

```
---
...
ootb_supply_chain_basic:
 gitops:
   ssh_secret: "SSH-SECRET-KEY"
...
```

To update the secret value, follow the instructions in Modify an AWS Secrets Manager secret.

When moving values, you must omit the `tap_install.values` root, but keep the remaining structure. All of the parent keys, for example, `ootb_supply_chain_basic.gitops` of the moved value, for example, `ssh_secret`, must be copied to the sensitive value YAML.

4. Commit and push the Tanzu Application Platform values:

   ```
   git add cluster-config/
   git commit -m "Configure initial values for TAP 1.5.0"
   git push
   ```

   Tanzu Sync fetches configuration from the hosted clone of the Git repository. For changes to take effect on the cluster, they must be pushed to that clone of the Git repository.

Deploy Tanzu Sync

Deploying Tanzu Sync kickstarts the GitOps workflow that initiates the Tanzu Application Platform installation.

After deployed, Tanzu Sync periodically polls the Git repository for changes. The following deployment process is only required once per cluster:

1. Install the Carvel tools `kapp` and `ytt` onto your `$PATH`:

   ```
   sudo cp $HOME/tanzu-cluster-essentials/kapp /usr/local/bin/kapp
   sudo cp $HOME/tanzu-cluster-essentials/ytt /usr/local/bin/ytt
   ```

   This step is required to ensure the correct deployment of the `tanzu-sync` App.

2. Ensure the Kubernetes cluster context is set to the EKS cluster.

   1. List the existing contexts:

      ```
      kubectl config get-contexts
      ```

   2. Set the context to the cluster that you want to deploy:

      ```
      kubectl config use-context CONTEXT-NAME
      ```

      Where `CONTEXT-NAME` can be retrieved from the outputs of the previous step.

3. Bootstrap the deployment.

   External Secrets Operator is installed from the package included in the Tanzu Application Plaform package repository. That repository must be fetched from the OCI registry initially.

   1. Set the following environment variables:

      ```
      export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
      export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
      ```

```
export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
```

Where:

- `MY-REGISTRY` is your container registry.
- `MY-REGISTRY-USER` is the user with read access to `MY-REGISTRY`.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.

2. Create a secret containing credentials to fetch from that OCI registry by using the provided script:

```
tanzu-sync/scripts/bootstrap.sh
```

These credentials are used exactly once to install the External Secrets Operator (ESO) package.

4. Install Tanzu Sync and start the GitOps workflow by deploying it to the cluster using `kapp` and `ytt`.

```
tanzu-sync/scripts/deploy.sh
```

Depending on the profile and components included, it may take 5-10 minutes for the Tanzu Application Platform to install. During this time, `kapp` waits for the deployment of Tanzu Sync to reconcile successfully. This is normal.

You can track the progress of the installation by watching the installation of those packages in a separate terminal window:

```
watch kubectl get pkgi -n tap-install
```

Install Tanzu Application Platform through Gitops with Secrets OPerationS (SOPS)

This topic tells you how to install Tanzu Application Platform (commonly known as TAP) through GitOps with secrets managed in a Git repository.

Caution

- Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.
- Tanzu GitOps Reference Implementation (RI) does not support changing the secrets management strategy for a cluster.

Prerequisites

Before installing Tanzu Application Platform, you need:

- **SOPS CLI** to view and edit SOPS encrypted files. To install the SOPS CLI, see SOPS documentation in GitHub.
- **Age CLI** to create an encryption key used to encrypt and decrypt sensitive data. To install the Age CLI, see age documentation in GitHub.
- Completed the Prerequisites.

- Accepted Tanzu Application Platform EULA and installed Tanzu CLI with any required plug-ins.
- Installed Cluster Essentials for Tanzu.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. If you don't relocate the images, Tanzu Application Platform depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

The supported registries are Harbor, Azure Container Registry, Google Container Registry, and Quay.io. See the following documentation for a registry to learn how to set it up:

- Harbor documentation
- Google Container Registry documentation
- Quay.io documentation

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

   ```
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
   export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
   export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export TAP_VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own container registry.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`.
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.
   - `TARGET-REPOSITORY` is your target repository, a folder or repository on `MY-REGISTRY` that serves as the location for the installation files for Tanzu Application Platform.

   VMware recommends using a JSON key file to authenticate with Google Container Registry. In this case, the value of `INSTALL_REGISTRY_USERNAME` is `_json_key` and the value of `INSTALL_REGISTRY_PASSWORD` is the content of the JSON key file. For more information about how to generate the JSON key file, see Google Container Registry documentation.

2. Install the Carvel tool `imgpkg` CLI.

   To query for the available versions of Tanzu Application Platform on VMWare Tanzu Network Registry, run:

```
imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/tap-pac
kages | sort -V
```

3. Relocate the images with the `imgpkg` CLI by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/tap-package
s:${TAP_VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-pac
kages
```

(Optional) Install Tanzu Application Platform in an air-gapped environment

Complete the following steps if you install Tanzu Application Platform in an air-gapped environment:

1. Relocate the Tanzu Build Service images to your registry:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
ps-package-repo:VERSION --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/
full-tbs-deps-package-repo
```

   Where:

   - `VERSION` is the version of Tanzu Build Service. You can retrieve this value by running `kubectl get package -n tap-install | grep buildservice`

2. Host a `grype` database in the air-gapped environment. For more information, see Use Grype in offline and air-gapped environments.

Create a new Git repository

1. In a hosted Git service, for example, GitHub or GitLab, create a new repository.

   This version of Tanzu GitOps RI only supports authenticating to a hosted Git repository by using SSH.

2. Initialize a new Git repository:

```
mkdir -p $HOME/tap-gitops
cd $HOME/tap-gitops

git init
git remote add origin git@github.com:my-organization/tap-gitops.git
```

3. Create a read-only deploy key for this new repository (recommended) or SSH key for an account with read access to this repository.

   The private portion of this key is referred to as `GIT_SSH_PRIVATE_KEY`.

Download and unpack Tanzu GitOps Reference Implementation (RI)

1. Sign in to VMware Tanzu Network.
2. Go to the Tanzu Application Platform product page.
3. Select **Release 1.5.12** from the release drop-down menu.
4. Click **Tanzu GitOps Reference Implementation**.

5. Unpack the downloaded TGZ file into the `$HOME/tap-gitops` directory by running:

```
tar xvf tanzu-gitops-ri-*.tgz -C $HOME/tap-gitops
```

6. Commit the initial state:

```
cd $HOME/tap-gitops

git add . && git commit -m "Initialize Tanzu GitOps RI"
git push -u origin
```

Create cluster configuration

1. Seed configuration for a cluster using SOPS:

```
cd $HOME/tap-gitops

./setup-repo.sh CLUSTER-NAME sops
```

Where:

- `CLUSTER-NAME` the name of your cluster.
- `sops` selects the Secrets OPerationS-based secrets management variant.

Example:

```
cd $HOME/tap-gitops

./setup-repo.sh full-tap-cluster sops
Created cluster configuration in ./clusters/full-tap-cluster.
...
```

This script creates the directory `clusters/full-tap-cluster/` and copies in the configuration required to sync this Git repository with the cluster and installing Tanzu Application Platform.

2. Commit and push:

```
git add . && git commit -m "Add full-tap-cluster"
git push
```

Configure Tanzu Application Platform

Tanzu Sync Reference Implementation (RI) splits the values configuration of Tanzu Application Platform into two categories:

- Sensitive TAP values, for example, credentials, encryptions keys and so on.
- Non-sensitive TAP values, for example, packages to exclude, namespace configuration and so on.

The following sections describe how to create these values files.

Preparing sensitive Tanzu Application Platform values

1. Generate Age public or secrets keys:

Note

Skip this step if you already have an Age key to encrypt or decrypt secrets.

```
mkdir -p $HOME/tmp-enc
chmod 700 $HOME/tmp-enc
cd $HOME/tmp-enc

age-keygen -o key.txt

cat key.txt
# created: 2023-02-08T10:55:35-07:00
# public key: age1ql3z7hjy54pw3hyww5ayyfg7zqgvc7w3j2elw8zmrj2kg5sfn9aqmcac8p
AGE-SECRET-KEY-my-secret-key
```

2. Create a plain YAML file `tap-sensitive-values.yaml` that contains a placeholder for the sensitive portion of Tanzu Application Platform values:

```
---
tap_install:
  sensitive_values:
```

3. Encrypt `tap-sensitive-values.yaml` with Age using SOPS:

```
export SOPS_AGE_RECIPIENTS=$(cat key.txt | grep "# public key: " | sed 's/# pub
lic key: //')
sops --encrypt tap-sensitive-values.yaml > tap-sensitive-values.sops.yaml
```

Where:

- `grep` is used to find the line containing the public key portion of the generated secret.
- `sed` is used to extract the public key from the line found by `grep`.

4. (Optional) Verify the encrypted file can be decrypted:

```
export SOPS_AGE_KEY_FILE=key.txt
sops --decrypt tap-sensitive-values.sops.yaml
```

(Optional) Verify the encrypted file can be edited directly by using SOPS:

```
sops tap-sensitive-values.sops.yaml
```

5. Move the sensitive Tanzu Application Platform values into the cluster config:

```
mv tap-sensitive-values.sops.yaml <GIT-REPO-ROOT>/clusters/<CLUSTER-NAME>/clust
er-config/values/
```

Example:

```
mv tap-sensitive-values.sops.yaml $HOME/tap-gitops/clusters/full-tap-cluster/cl
uster-config/values/
```

6. (Optional) Retain the Age identity key file in a safe and secure place such as a password manager, and purge the scratch space:

```
mv key.txt SAFE-LOCATION/
export SOPS_AGE_KEY_FILE=SAFE-LOCATION/key.txt
rm -rf $HOME/tmp-enc
```

Preparing non-sensitive Tanzu Application Platform values

Create a plain YAML file `<GIT-REPO-ROOT>/clusters/<CLUSTER-NAME>/cluster-config/values/tap-non-sensitive-values.yaml` by using the Full Profile sample as a guide:

Example:

```
---
tap_install:
  values:
    ceip_policy_disclosed: true
    excluded_packages:
    - policy.apps.tanzu.vmware.com
...
```

To install Tanzu Application Service in an offline environment, you must configure `Tanzu Build Service` and `Grype` to work in an air-gapped environment:

```
---
tap_install:
  values:
    ...
    buildservice:
      exclude_dependencies: true
    grype:
      db:
        dbUpdateUrl: INTERNAL-VULN-DB-URL
```

Where:

- `INTERNAL-VULN-DB-URL` URL that points to the internal file server.

Updating sensitive Tanzu Application Platform values

After filling in the non-sensitive values, follow these steps to extract the sensitive values into `tap-sensitive-values.sops.yaml` that you prepared earlier:

1. Open an editor through SOPS to edit the encrypted sensitive values file:

   ```
   sops <GIT-REPO-ROOT>/clusters/<CLUSTER-NAME>/cluster-config/values/tap-sensitiv
   e-values.sops.yaml
   ```

   Example:

   ```
   sops $HOME/tap-gitops/clusters/full-tap-cluster/cluster-config/values/tap-sensi
   tive-values.sops.yaml
   ```

2. Add the sensitive values:

   Example of the container registry credentials using basic authentication:

   ```
   ---
   tap_install:
    sensitive_values:
      shared:
        image_registry:
          project_path: "example.com/my-project/tap"
          username: "my_username"
          password: "my_password"
   ```

   Example of the container registry credentials using Google Container Registry:

```
---
tap_install:
 sensitive_values:
  shared:
    image_registry:
       project_path: "gcr.io/my-project/tap"
       username: "_json_key"
       password: |
         {
           "type": "service_account",
           "project_id": "my-project",
           "private_key_id": "my-private-key-id",
           "private_key": "-----BEGIN PRIVATE KEY-----\n.........\n-----END PR
IVATE KEY-----\n",
           ...
         }
```

Generate Tanzu Application Platform installation and Tanzu Sync configuration

Follow these steps to generate the Tanzu Application Platform installation and Tanzu Sync configuration:

1. Set up environment variables by running:

   ```
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export GIT_SSH_PRIVATE_KEY=PRIVATE-KEY
   export GIT_KNOWN_HOSTS=KNOWN-HOST-LIST
   export SOPS_AGE_KEY=AGE-KEY
   export TAP_PKGR_REPO=TAP-PACKAGE-OCI-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY` is your container registry.
   - `MY-REGISTRY-USER` is the user with read access to `MY-REGISTRY`.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `PRIVATE-KEY` is the contents of an SSH private key file with read access to your Git repository.
   - `HOST-LIST` is the list of known hosts for Git host service.
   - `AGE-KEY` is the contents of the Age key generated earlier.
   - `TAP-PACKAGE-OCI-REPOSITORY` is the fully-qualified path to the OCI repository hosting the Tanzu Application Platform images. If the images are relocated as described in Relocate images to a registry, this value is `${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages`.

   Example of the Git repo hosted on GitHub:

   ```
   export INSTALL_REGISTRY_HOSTNAME=registry.tanzu.vmware.com
   export INSTALL_REGISTRY_USERNAME=foo@example.com
   export INSTALL_REGISTRY_PASSWORD=my-password
   export GIT_SSH_PRIVATE_KEY=$(cat $HOME/.ssh/my_private_key)
   export GIT_KNOWN_HOSTS=$(ssh-keyscan github.com)
   export SOPS_AGE_KEY=$(cat $HOME/key.txt)
   export TAP_PKGR_REPO=registry.tanzu.vmware.com/tanzu-application-platform/tap-p
   ackages
   ```

2. Generate the Tanzu Application Platform install and the Tanzu Sync configuration files by using the provided script:

```
cd <GIT-REPO-ROOT>/clusters/<CLUSTER-NAME>

./tanzu-sync/scripts/configure.sh
```

Example:

```
cd $HOME/tap-gitops/clusters/full-tap-cluster

./tanzu-sync/scripts/configure.sh
```

3. (Optional) Update Tanzu Application Platform to use the latest patch:

Update the Tanzu Application Platform version in `GIT-REPO-ROOT/clusters/CLUSTER-NAME/cluster-config/values/tap-install-values.yaml`:

```
tap_install:
    ...
    version:
        package_repo_bundle_tag: "1.5.6" # Populate these values with the lates
t patch version.
        package_version: "1.5.6"
```

Where:

- `package_repo_bundle_tag` is the version of Tanzu Application Platform you want to upgrade to.
- `package_version` is the version of Tanzu Application Platform you want to upgrade to. This version must match `package_repo_bundle_tag`.

Note

Tanzu GitOps RI does not provide a separate artifact for each patch version within a minor line. For example, Tanzu Application Platform v1.5.x contains the GitOps artifact with v1.5.0 only.

4. Commit the generated configured to Git repository:

```
git add cluster-config/ tanzu-sync/
git commit -m "Configure install of TAP 1.5.0"
git push
```

Deploy Tanzu Sync

1. Install the Carvel tools `kapp` and `ytt` onto your `$PATH`:

```
sudo cp $HOME/tanzu-cluster-essentials/kapp /usr/local/bin/kapp
sudo cp $HOME/tanzu-cluster-essentials/ytt /usr/local/bin/ytt
```

2. Set the Kubernetes cluster context.
    1. List the existing contexts:

    ```
    kubectl config get-contexts
    ```

    2. Set the context to the cluster that you want to deploy:

```
kubectl config use-context CONTEXT-NAME
```

Where `CONTEXT-NAME` can be retrieved from the outputs of the previous step.

3. Deploy the Tanzu Sync component:

```
cd GIT-REPO-ROOT/clusters/CLUSTER-NAME

./tanzu-sync/scripts/deploy.sh
```

Example:

```
cd $HOME/tap-gitops/clusters/full-tap-cluster

./tanzu-sync/scripts/deploy.sh
```

Note

Depending on the profile and components included, it may take 5-10 minutes for Tanzu Application Platform to install. During this time, `kapp` waits for the deployment of Tanzu Sync to reconcile successfully. This is normal.

Install individual packages

You can install Tanzu Application Platform (commonly known as TAP) through predefined profiles or through individual packages. Use this topic to learn how to install each individual package. For more information about installing through profiles, see Components and installation profiles.

Installing individual Tanzu Application Platform packages is useful if you do not want to use a profile to install packages or if you want to install additional packages after installing a profile. Before installing the packages, be sure to complete the prerequisites, configure and verify the cluster, accept the EULA, and install the Tanzu CLI with any required plug-ins. For more information, see Prerequisites.

Install pages for individual Tanzu Application Platform packages

- Install API Auto Registration
- Install API portal
- Install Application Accelerator
- Install Application Configuration Service
- Install Application Live View
- Install Application Single Sign-On
- Install Bitnami Services
- Install cert-manager
- Install Cloud Native Runtimes
- Install Contour
- Install Crossplane
- Install default roles for Tanzu Application Platform

- Install Developer Conventions
- Install Eventing
- Install Flux CD Source Controller
- Install Learning Center for Tanzu Application Platform
- Install Out of the Box Templates
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Service Bindings
- Install Services Toolkit
- Install Source Controller
- Install Spring Boot conventions
- Install Supply Chain Choreographer
- Install Supply Chain Security Tools - Store
- Install Supply Chain Security Tools - Policy Controller
- Install Supply Chain Security Tools - Scan
- Install Tanzu Application Platform GUI
- Install Tanzu Build Service
- Install Tekton
- Install Telemetry

Verify the installed packages

Use the following procedure to verify that the packages are installed.

1. List the installed packages by running:

   ```
   tanzu package installed list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package installed list --namespace tap-install
   \ Retrieving installed packages...
   NAME                     PACKAGE-NAME                                    PAC
   KAGE-VERSION  STATUS
   api-portal               api-portal.tanzu.vmware.com                     1.
   0.3           Reconcile succeeded
   app-accelerator          accelerator.apps.tanzu.vmware.com               1.
   0.0           Reconcile succeeded
   app-live-view            appliveview.tanzu.vmware.com                    1.
   0.2           Reconcile succeeded
   appliveview-conventions  build.appliveview.tanzu.vmware.com              1.
   0.2           Reconcile succeeded
   cartographer             cartographer.tanzu.vmware.com                   0.
   1.0           Reconcile succeeded
   cloud-native-runtimes    cnrs.tanzu.vmware.com                           1.
   0.3           Reconcile succeeded
   convention-controller    controller.conventions.apps.tanzu.vmware.com    0.
   7.0           Reconcile succeeded
   developer-conventions    developer-conventions.tanzu.vmware.com          0.
   3.0-build.1   Reconcile succeeded
   grype-scanner            grype.scanning.apps.tanzu.vmware.com            1.
   ```

```
0.0            Reconcile succeeded
image-policy-webhook     image-policy-webhook.signing.apps.tanzu.vmware.com 1.
1.2            Reconcile succeeded
metadata-store           metadata-store.apps.tanzu.vmware.com               1.
0.2            Reconcile succeeded
ootb-supply-chain-basic  ootb-supply-chain-basic.tanzu.vmware.com          0.
5.1            Reconcile succeeded
ootb-templates           ootb-templates.tanzu.vmware.com                   0.
5.1            Reconcile succeeded
scan-controller          scanning.apps.tanzu.vmware.com                    1.
0.0            Reconcile succeeded
service-bindings         service-bindings.labs.vmware.com                  0.
5.0            Reconcile succeeded
services-toolkit         services-toolkit.tanzu.vmware.com                 0.
8.0            Reconcile succeeded
source-controller        controller.source.apps.tanzu.vmware.com           0.
2.0            Reconcile succeeded
sso4k8s-install          sso.apps.tanzu.vmware.com                         1.
0.0-beta.2-31  Reconcile succeeded
tap-gui                  tap-gui.tanzu.vmware.com                          0.
3.0-rc.4       Reconcile succeeded
tekton-pipelines         tekton.tanzu.vmware.com                           0.3
0.0           Reconcile succeeded
tbs                      buildservice.tanzu.vmware.com                     1.
5.0            Reconcile succeeded
```

Next steps

- Set up developer namespaces to use your installed packages

Set up developer namespaces to use your installed packages

For details about how to automatically set up your developer namespaces, see Provision developer namespaces in Namespace Provisioner.

Additional configuration for testing and scanning

If you plan to install or have already installed Out of the Box Supply Chains with Testing and Scanning, you can use Namespace Provisioner to set up the required resources. For more information, see Customize installation in the Namespace Provisioner documentation for configuration steps.

Legacy namespace setup

To use the legacy manual process for setting up developer namespaces, see Legacy namespace setup.

Next steps

- Install Tanzu Developer Tools for your VS Code

Install Tanzu Developer Tools for your VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

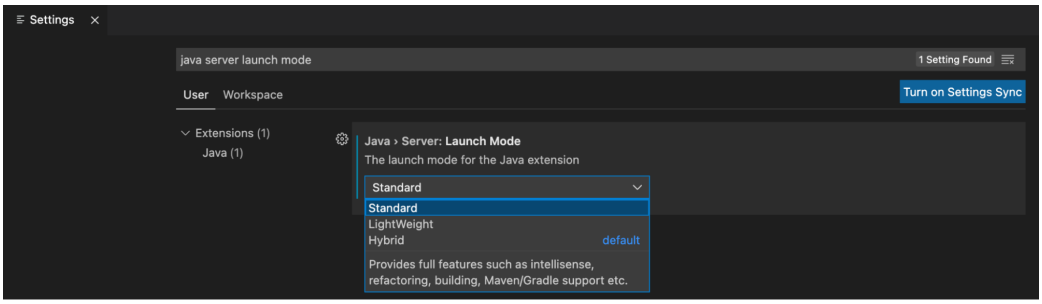
Install

To install the extension:

1. Sign in to VMware Tanzu Network and download Tanzu Developer Tools for Visual Studio Code.
2. Open VS Code.
3. Press cmd+shift+P to open the Command Palette and run `Extensions: Install from VSIX...`.

4. Select the extension file **tanzu-vscode-extension.vsix**.
5. If you do not have the following extensions, and they do not automatically install, install them from VS Code Marketplace:
   - Debugger for Java
   - Language Support for Java(™) by Red Hat
   - YAML
6. Ensure Language Support for Java is running in Standard Mode. You can configure it in the **Settings** menu by going to **Code** > **Preferences** > **Settings** under **Java > Server: Launch Mode**.

   When the JDK and Language Support for Java are configured correctly, you see that the integrated development environment creates a directory target where the code is compiled.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: For more information, see Integrating Live Hover by using Spring Boot Tools. Reload VS Code for this change to take effect.
    - **Source Image**: (Required) The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name.
    - **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
    - **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in `kubeconfig` is the default.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

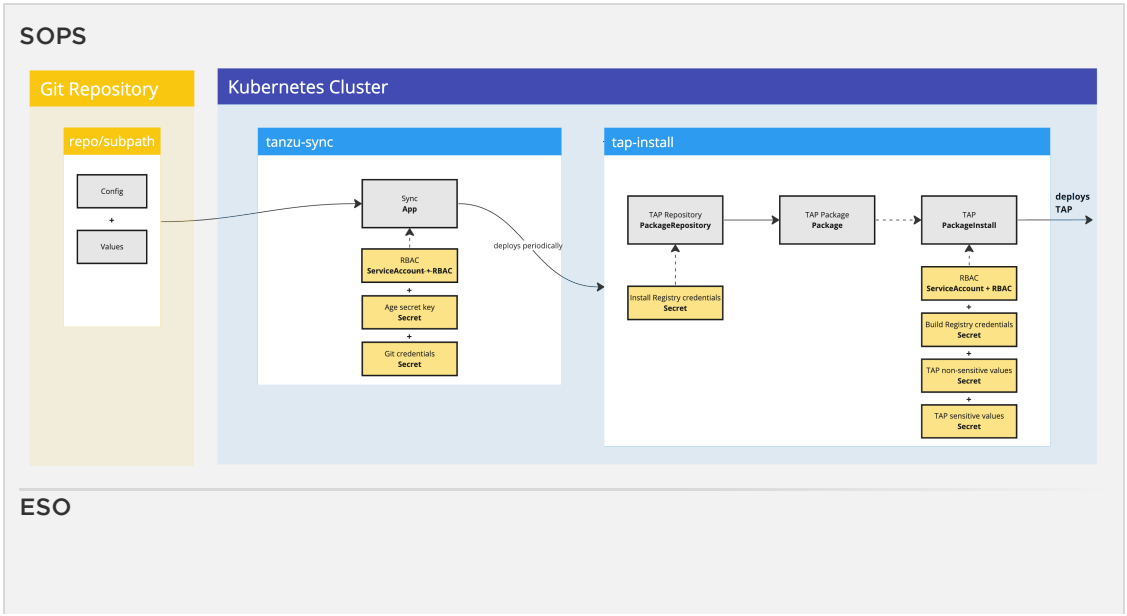
1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and select **Uninstall**.

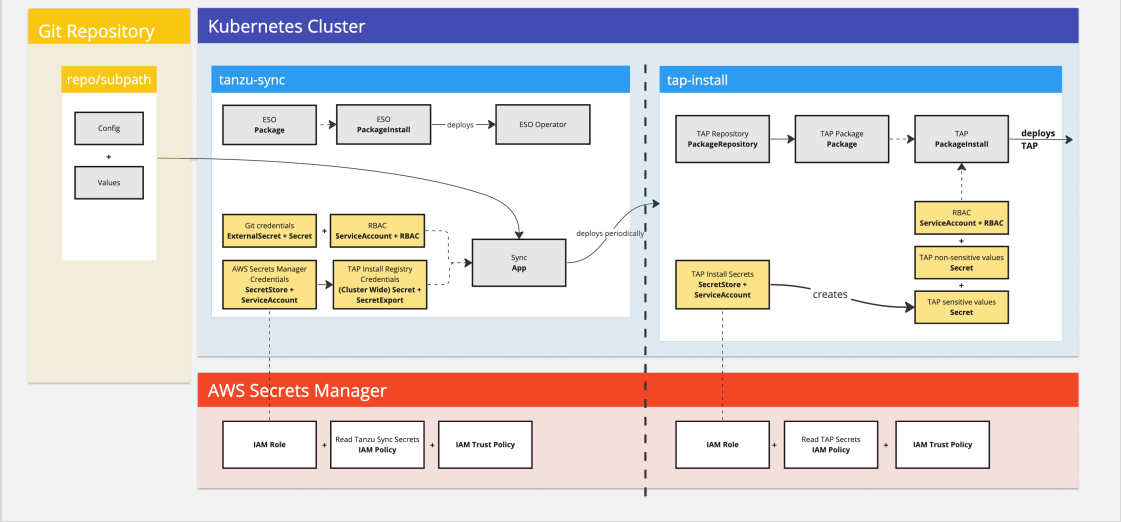
Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Tanzu GitOps RI Reference Documentation

The following diagrams shows you the components that are installed as part of Tanzu GitOps Reference Implementation (RI) and how they work together to automate the installation of Tanzu Application Platform (commonly known as TAP):





Caution

Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

Tanzu Sync Carvel Application

Tanzu Sync consists of a Carvel application named `sync` that is installed in the `tanzu-sync` namespace. The sync application:

1. Fetches a Git repository that contains configuration for Tanzu Application Platform.
2. Templates with `ytt` a set of resources and data values.
3. Deploys with `kapp` a set of resources to install Tanzu Application Platform, with any other user specified configuration in the Git Repository.

Choosing SOPS or ESO

The following table outlines the Kubernetes distributions and secret management solutions that SOPS and ESO support:

Choose	IaaS	Secrets Manager
SOPS	Any TAP supported IaaS	N/A
ESO	AWS (EKS)	AWS Secrets Manager

Note

Future release will include additional Secrets Managers for ESO.

The following table describes a few common use cases and scenarios for SOPS and ESO:

I want ...	SOPS	ESO
Sensitive data encrypted inside the Git repository.		
Sensitive data to be stored outside the Git repository.		
Minimal setup. No external secret storage system		

I want ...	SOPS	ESO
To manage sensitive data myself. For example, storing keys, rotation and usage auditing.)		
To utilize sensitive data management. For example, storage, rotation and usage auditing by a third-party solution.		

Git Repository structure

Tanzu Sync Application fetches our deployable content from a Git repository that must match the following structure:

Git repository for a cluster named `full-tap-cluster`:

```
├── .catalog
│   ├── tanzu-sync
│   │   └── 0.0.3
│   └── tap-install
│       └── 1.5.0
├── README.md
├── clusters
│   └── full-tap-cluster
│       ├── README.md
│       ├── cluster-config
│       │   ├── config
│       │   │   └── tap-install
│       │   │       └── .tanzu-managed
│       │   └── values
│       └── tanzu-sync
│           ├── app
│           │   ├── config
│           │   │   └── .tanzu-managed
│           │   └── values
│           ├── bootstrap
│           └── scripts
└── setup-repo.sh
```

Where:

- `.catalog`: VMware supplied directory of resources and configuration to install Tanzu Sync and Tanzu Application Platform.
    - `tanzu-sync`: Contains the Carvel Packaging App which supports a GitOps workflow for fetching, templating and deploying the `clusters/full-tap-cluster/cluster-config` directory of this repository.
    - `tap-install`: Contains the configuration to install Tanzu Application Platform.
- `clusters/full-tap-cluster`
    - `cluster-config`
        - `config`: Contains the Tanzu Application Platform installation configuration. This directory can be extended to include any desired resources managed through GitOps to your cluster.
        - `.tanzu-managed`: Contains VMware managed Kubernetes resource files to install Tanzu Application Platform. Do not alter this value.
        - `values`: Contains the plain YAML data files which configure the application.
    - `tanzu-sync`
        - `app`: Contains the main Carvel Packaging App that runs on the cluster. It fetches, templates and deploys your Tanzu Application Platform installation from `clusters/full-tap-cluster/cluster-config`.

- `bootstrap`: Contains secret provider specific bootstrapping if required.
- `scripts`: Contains helper scripts to assist with the configuration and deployment of Tanzu GitOps RI.

Configuration of Tanzu Sync without helper scripts

1. The following plain YAML values files are required to run Tanzu Sync:
    - Tanzu Sync App:

      `clusters/full-tap-cluster/tanzu-sync/app/values/values.yaml` adhering to the following schema:

      ```
      #@data/values-schema
      #@overlay/match-child-defaults missing_ok=True
      ---
      git:
        url: ""
        ref: ""
        sub_path: ""

      tap_package_repository:
        oci_repository: ""
      ```

      Example:

      ```
      ---
      git:
        url: git@github.com:my-org/gitops-tap.git
        ref: origin/main
        sub_path: clusters/full-tap-cluster/cluster-config

      tap_package_repository:
        oci_repository: registry.example.com/tanzu-application-platform/tap-pac
      kages
      ```

    - Tanzu Application Platform Install:

      `clusters/full-tap-cluster/cluster-config/config/values/install-values.yaml` adhering to the following schema:

      ```
      #@data/values-schema
      #@overlay/match-child-defaults missing_ok=True
      ---
      tap_install:
        package_repository:
          oci_repository: ""
        #@schema/type any=True
        values: {}
      ```

      Example:

      ```
      tap_install:
        package_repository:
          oci_repository: registry.example.com/tanzu-application-platform/tap-p
      ackages
        values:
          shared:
            ingress_domain: example.vmware.com
          ceip_policy_disclosed: true
      ```

`clusters/full-tap-cluster/cluster-config/config/values/sensitive-values.sops.yaml` adhering to the following schema:

```
#@data/values-schema
#@overlay/match-child-defaults missing_ok=True
---
tap_install:
  #@schema/nullable
  #@schema/validation not_null=True
  #@schema/type any=True
  sensitive_values: {}
```

Example:

```
tap_install:
  sensitive_values:
    shared:
      image_registry:
        project_path: example.registry.com/my-project/my-user/tap
        username: my-username
        password: my-password
```

2. The following is used to deploy the application by using `kapp`:

```
kapp deploy --app tanzu-sync --file <(ytt \
    --file tanzu-sync/app/config \
    --file cluster-config/config/tap-install/.tanzu-managed/version.yaml \
    --data-values-file tanzu-sync/app/values/ \
    --data-value secrets.sops.age_key=$(cat $HOME/key.txt) \
    --data-value secrets.sops.registry.hostname="hostname" \
    --data-value secrets.sops.registry.username="foo@example.com" \
    --data-value secrets.sops.registry.password="password" \
    --data-value secrets.sops.git.ssh.private_key=$(cat $HOME/.ssh/my_private_k
ey) \
    --data-value secrets.sops.git.ssh.known_hosts=$(ssh-keyscan github.com) \
)
```

Tanzu Sync Scripts

> **Caution**
>
> The provided scripts are intended to help set up your Git repository to work with a GitOps approach, they are subject to change or removal between releases.

VMware provides a set of convenience bash scripts in `clusters/MY-CLUSTER/tanzu-sync/scripts` to help you set up your Git repository and configure the values as described in the previous section:

- `setup-repo.sh`: Populates a Git repository with the structure described in the Git Repository structure section.
- `configure.sh`: Generates the values files described in the Configuration of values without helper scripts section.
- `deploy.sh`: A light wrapper around a simple `kapp deploy` given the data values from the previous section and sensitive values which must not be stored on disk.

Customize your package installation

You can customize your package configuration that is not exposed through data values by using annotations and ytt overlays.

You can customize a package that was installed manually or that was installed by using a Tanzu Application Platform profile.

Customize a package that was manually installed

To customize a package that was installed manually:

1. Create a `secret.yml` file with a `Secret` that contains your ytt overlay. For example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
    name: tap-overlay
    namespace: tap-install
   stringData:
    custom-package-overlay.yml: |
      CUSTOM-OVERLAY
   ```

   For more information about ytt overlays, see the Carvel documentation.

2. Apply the `Secret` to your cluster by running:

   ```
   kubectl apply -f secret.yml
   ```

3. Update your `PackageInstall` to include the `ext.packaging.carvel.dev/ytt-paths-from-secret-name.x` annotation to reference your new overlay `Secret`. For example:

   ```
   apiVersion: packaging.carvel.dev/v1alpha1
   kind: PackageInstall
   metadata:
    name: PACKAGE-NAME
    namespace: tap-install
    annotations:
      ext.packaging.carvel.dev/ytt-paths-from-secret-name: tap-overlay
   ...
   ```

   Note

   You can suffix the extension annotation with `.x`, where `x` is a number, to apply multiple overlays. For more information, see the Carvel documentation.

Customize a package that was installed by using a profile

To add an overlay to a package that was installed by using a Tanzu Application Platform profile:

1. Create a `Secret` with your ytt overlay. For more information about ytt overlays, see the Carvel documentation.

2. Update your values file to include a `package_overlays` field:

   ```
   package_overlays:
   - name: PACKAGE-NAME
     secrets:
     - name: SECRET-NAME
   ```

Where `PACKAGE-NAME` is the target package for the overlay. For example, `tap-gui`.

3. Update Tanzu Application Platform by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v 1.5.12  --values-
file tap-values.yaml -n tap-install
```

For information about Tanzu Application Platform profiles, see Installing Tanzu Application Platform package and profiles.

Upgrade your Tanzu Application Platform

This document tells you how to upgrade your Tanzu Application Platform (commonly known as TAP).

You can perform a fresh install of Tanzu Application Platform by following the instructions in Installing Tanzu Application Platform.

Prerequisites

Before you upgrade Tanzu Application Platform:

- Verify that you meet all the prerequisites of the target Tanzu Application Platform version. If the target Tanzu Application Platform version does not support your existing Kubernetes version, VMware recommends upgrading to a supported version before proceeding with the upgrade.
- For information about installing your Tanzu Application Platform, see Install your Tanzu Application Platform profile.
- Ensure that Tanzu CLI is updated to the version recommended by the target Tanzu Application Platform version. For information about installing or updating the Tanzu CLI and plug-ins, see Install or update the Tanzu CLI and plug-ins.
- For information about Tanzu Application Platform GUI considerations, see Tanzu Application Platform GUI Considerations.
- Verify all packages are reconciled by running `tanzu package installed list -A`.
- To avoid the temporary warning state that is described in Update the new package repository, upgrade to Cluster Essentials v1.5. See Cluster Essentials documentation for more information about the upgrade procedures.

Update the new package repository

Follow these steps to update the new package repository:

1. Relocate the latest version of Tanzu Application Platform images by following step 1 through step 6 in Relocate images to a registry.

Important

Make sure to update the `TAP_VERSION` to the target version of Tanzu Application Platform you are migrating to. For example, `1.5.12`.

2. Add the target version of the Tanzu Application Platform package repository by running:

Cluster Essentials 1.2 or above

```
tanzu package repository add tanzu-tap-repository \
--url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tap-packages:$TAP_VERSION
\
--namespace tap-install
```

Cluster Essentials 1.1 or 1.0

```
tanzu package repository update tanzu-tap-repository \
--url ${INSTALL_REGISTRY_HOSTNAME}/TARGET-REPOSITORY/tap-packages:${TAP_VERSI
ON} \
--namespace tap-install
```

Expect to see the installed Tanzu Application Platform packages in a temporary “Reconcile Failed” state, following a “Package not found” warning. These warnings will disappear after you upgrade the installed Tanzu Application Platform packages to version 1.2.0.

3. Verify you have added the new package repository by running:

   ```
   tanzu package repository get TAP-REPO-NAME --namespace tap-install
   ```

   Where `TAP-REPO-NAME` is the package repository name. It must match with either `NEW-TANZU-TAP-REPOSITORY` or `tanzu-tap-repository` in the previous step.

Perform the upgrade of Tanzu Application Platform

The following sections describe how to upgrade in different scenarios.

Upgrade instructions for Profile-based installation

The following changes affect the upgrade procedures:

- **Keyless support deactivated by default**

  In Tanzu Application Platform v1.5.0, keyless support is deactivated by default. For more information, see Install Supply Chain Security Tools - Policy Controller.

  To support the keyless authorities in `ClusterImagePolicy`, Policy Controller no longer initializes TUF by default. To continue using keyless authorities, you must set the `policy.tuf_enabled` field to `true` in the `tap-values.yaml` file during the upgrade process.

  By default, the public official Sigstore “The Update Framework (TUF) server” is used. You can use an alternative Sigstore Stack by setting `policy.tuf_mirror` and `policy.tuf_root`.

- **Image Policy Webhook no longer in use**

  Tanzu Application Platform v1.5.0 removes Image Policy Webhook. If you use Image Policy Webhook in the previous version of Tanzu Application Platform, you must migrate the `ClusterImagePolicy` resource from Image Policy Webhook to Policy Controller. For more information, see Migration From Supply Chain Security Tools - Sign.

- **CVE results require a read-write service account**

  Tanzu Application Platform v1.3.0 uses a read-only service account. In Tanzu Application Platform v1.4.0 and later, enabling CVE results for the Supply Chain Choreographer and Security Analysis GUI plug-ins requires a read-write service account. For more information, see Enable CVE scan results.

If you installed Tanzu Application Platform by using a profile, you can perform the upgrade by running the following command in the directory where the `tap-values.yaml` file resides:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP_VERSION}  --values
-file tap-values.yaml -n tap-install
```

When upgrading to Tanzu Application Platform v1.5, you might encounter a temporary resource reconciliation failure. This error does not persist and the packages will reconcile subsequently. To facilitate the reconciliation of packages, you can execute the `tanzu package installed kick -n tap-install tap -y` command repeatedly.

Upgrade the full dependencies package

If you installed the full dependencies package, you can upgrade the package by following these steps:

1. After upgrading Tanzu Application Platform, retrieve the latest version of the Tanzu Build Service package by running:

   ```
   tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
   all
   ```

2. Relocate the Tanzu Build Service `full` dependencies package repository by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
   ps-package-repo:VERSION \
   --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tbs-full-deps
   ```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.

3. Update the Tanzu Build Service `full` dependencies package repository by running:

   ```
   tanzu package repository add tbs-full-deps-repository \
     --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/tbs-full-deps:VERSION \
     --namespace tap-install
   ```

4. Update the `full` dependencies package by running:

   ```
   tanzu package installed update full-tbs-deps -p full-tbs-deps.tanzu.vmware.com
   -v VERSION -n tap-install
   ```

Multicluster upgrade order

Upgrading a multicluster deployment requires updating multiple clusters with different profiles. If upgrades are not performed at the exact same time, different clusters have different versions of profiles installed temporarily. This might cause a temporary API mismatch that leads to errors. Those errors eventually disappear when the versions are consistent across all clusters.

To reduce the likelihood of temporary failures, follow these steps to upgrade your multicluster deployment:

1. Upgrade the view-profile cluster.
2. Upgrade the remaining clusters in any order.

Upgrade instructions for component-specific installation

For information about upgrading Tanzu Application Platform GUI, see Upgrade Tanzu Application Platform GUI. For information about upgrading Supply Chain Security Tools - Scan, see Upgrade Supply Chain Security Tools - Scan.

Verify the upgrade

Verify the versions of packages after the upgrade by running:

```
tanzu package installed list --namespace tap-install
```

Your output is similar, but probably not identical, to the following example output:

```
- Retrieving installed packages...
  NAME                                PACKAGE-NAME
PACKAGE-VERSION  STATUS
  accelerator                         accelerator.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
  api-auto-registration               apis.apps.tanzu.vmware.com
0.1.1            Reconcile succeeded
  api-portal                          api-portal.tanzu.vmware.com
1.2.2            Reconcile succeeded
  appliveview                         backend.appliveview.tanzu.vmware.com
1.3.0            Reconcile succeeded
  appliveview-connector               connector.appliveview.tanzu.vmware.com
1.3.0            Reconcile succeeded
  appliveview-conventions             conventions.appliveview.tanzu.vmware.com
1.3.0            Reconcile succeeded
  appsso                              sso.apps.tanzu.vmware.com
2.0.0            Reconcile succeeded
  buildservice                        buildservice.tanzu.vmware.com
1.7.1            Reconcile succeeded
  cartographer                        cartographer.tanzu.vmware.com
0.5.3            Reconcile succeeded
  cert-manager                        cert-manager.tanzu.vmware.com
1.7.2+tap.1      Reconcile succeeded
  cnrs                                cnrs.tanzu.vmware.com
2.0.1            Reconcile succeeded
  contour                             contour.tanzu.vmware.com
1.22.0+tap.3     Reconcile succeeded
  conventions-controller              controller.conventions.apps.tanzu.vmware.com
0.7.1            Reconcile succeeded
  developer-conventions               developer-conventions.tanzu.vmware.com
0.8.0            Reconcile succeeded
  eventing                            eventing.tanzu.vmware.com
2.0.1            Reconcile succeeded
  fluxcd-source-controller            fluxcd.source.controller.tanzu.vmware.com
0.27.0+tap.1     Reconcile succeeded
  grype                               grype.scanning.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
  image-policy-webhook                image-policy-webhook.signing.apps.tanzu.vmware.c
om   1.1.7            Reconcile succeeded
  learningcenter                      learningcenter.tanzu.vmware.com
0.2.3            Reconcile succeeded
  learningcenter-workshops            workshops.learningcenter.tanzu.vmware.com
0.2.2            Reconcile succeeded
  metadata-store                      metadata-store.apps.tanzu.vmware.com
1.3.3            Reconcile succeeded
  ootb-delivery-basic                 ootb-delivery-basic.tanzu.vmware.com
0.10.2           Reconcile succeeded
  ootb-supply-chain-testing-scanning  ootb-supply-chain-testing-scanning.tanzu.vmware.
com  0.10.2           Reconcile succeeded
  ootb-templates                      ootb-templates.tanzu.vmware.com
0.10.2           Reconcile succeeded
  policy-controller                   policy.apps.tanzu.vmware.com
1.1.1            Reconcile succeeded
  scanning                            scanning.apps.tanzu.vmware.com
1.3.0            Reconcile succeeded
  service-bindings                    service-bindings.labs.vmware.com
0.8.0            Reconcile succeeded
```

```
  services-toolkit                    services-toolkit.tanzu.vmware.com
0.8.0             Reconcile succeeded
  source-controller                   controller.source.apps.tanzu.vmware.com
0.5.0             Reconcile succeeded
  spring-boot-conventions             spring-boot-conventions.tanzu.vmware.com
0.5.0             Reconcile succeeded
  tap                                 tap.tanzu.vmware.com
1.3.0             Reconcile succeeded
  tap-auth                            tap-auth.tanzu.vmware.com
1.1.0             Reconcile succeeded
  tap-gui                             tap-gui.tanzu.vmware.com
1.3.0             Reconcile succeeded
  tap-telemetry                       tap-telemetry.tanzu.vmware.com
0.3.1             Reconcile succeeded
  tekton-pipelines                    tekton.tanzu.vmware.com
0.39.0+tap.2      Reconcile succeeded
```

Opt out of telemetry collection

This topic tells you how to opt out of the VMware Customer Experience Improvement Program (CEIP) and out of Pendo telemetry on an organizational level.

There are two components for telemetry collection in Tanzu Application Platform (commonly known as TAP) under the VMware Customer Experience Improvement Program (CEIP):

1. The standard CEIP telemetry collection
2. Pendo telemetry from Tanzu Application Platform GUI

Each telemetry component has its own opt-in and opt-out process. The CEIP telemetry opt-out decision can be made at an organizational level, whereas the decision regarding the Pendo telemetry is available both on an organizational level and at an individual user level.

When you install Tanzu Application Platform, both standard CEIP and Pendo telemetry are turned on by default. If you opt out of standard CEIP telemetry collection, VMware cannot offer you proactive support and the other benefits that accompany participation in the CEIP.

Turn off standard CEIP telemetry collection

To deactivate Pendo telemetry collection, see Enable or deactivate the Pendo telemetry for the organization later in the topic.

Note

If you decide to opt in to Pendo telemetry collection, each user is given the option to opt in or opt out. For more information, see Opt in or opt out of Pendo telemetry for Tanzu Application Platform GUI.

To turn off CEIP telemetry collection, follow these instructions:

kubectl

To turn off telemetry collection on Tanzu Application Platform by using kubectl:

1. Ensure that your Kubernetes context is pointing to the cluster where Tanzu Application Platform is installed.
2. Run:

```
kubectl apply -f - <<EOF
apiVersion: v1
```

```
kind: Namespace
metadata:
  name: vmware-system-telemetry
---
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: vmware-system-telemetry
  name: vmware-telemetry-cluster-ceip
data:
  level: disabled
EOF
```

3. If you already have Tanzu Application Platform installed, restart the telemetry collector to apply the change:

   ```
   kubectl delete pods --namespace tap-telemetry --all
   ```

Your Tanzu Application Platform deployment now no longer emits telemetry, and you have opted out of the CEIP.

Tanzu CLI

The Tanzu CLI provides a telemetry plug-in enabled by the Tanzu Framework v0.25.0, which is included in Tanzu Application Platform v1.3 and later.

To turn off telemetry collection on your Tanzu Application Platform by using the Tanzu CLI, run:

```
tanzu telemetry update --CEIP-opt-out
```

To learn more about how to update the telemetry settings, run:

```
tanzu telemetry update --help
```

Your Tanzu Application Platform deployment now no longer emits telemetry, and you have opted out of the CEIP.

Turn off Pendo telemetry collection

To deactivate the program for the entire organization, add the following parameters to your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    pendoAnalytics:
      enabled: false
```

To enable Pendo telemetry for the organization, add the following parameters to your `tap-values.yaml` file:

```
tap_gui:
  app_config:
    pendoAnalytics:
      enabled: true
```

Opt in or opt out of Pendo telemetry for Tanzu Application Platform GUI

Tanzu Application Platform GUI (commonly called TAP GUI) uses Pendo.io to better understand the way users interact with it to provide a better user experience for VMware customers and to improve VMware products and services.

Pendo.io collects data based on your interaction with the software, such as clickstream data and page loads, hashed user ID, and limited browser and device information.

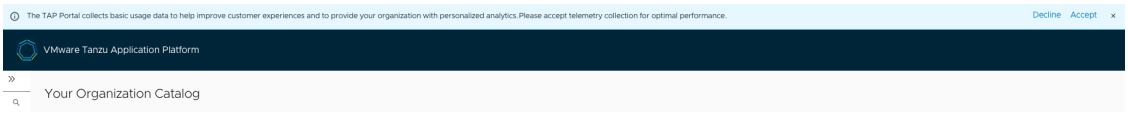
To enable or deactivate Pendo telemetry for the organization, see Enable or deactivate the Pendo telemetry for the organization.

Note

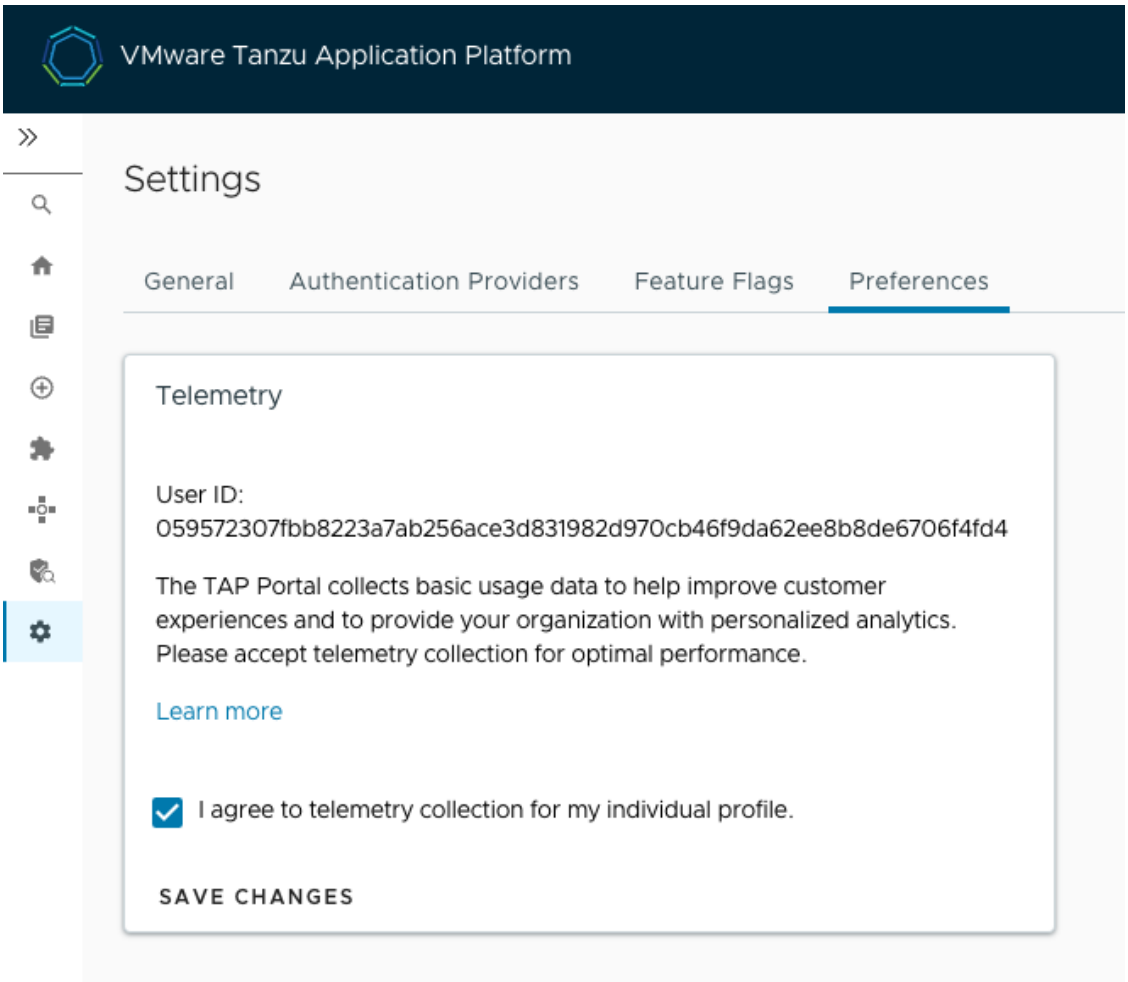
Pendo telemetry is separate from the VMware CEIP telemetry. There is a separate process for opting in or out of the VMware CEIP. For more information, see Opt out of telemetry collection.

Opt in or opt out of Pendo telemetry from Tanzu Application Platform GUI

After the Pendo telemetry is enabled for the organization, in accordance with VMware policy each user is prompted to agree to participate in the program or decline.



Each individual's preference is stored in Tanzu Application Platform GUI and can be modified at any time. To change your preferences, go to **Settings** > **Preferences**.



Request to delete your anonymized data

If you no longer want to participate in the program and you want VMware to delete all your anonymized data, please send an email requesting deletion, with your hashed User ID, to tap-pendo@groups.vmware.com.

This enables VMware to identify your anonymized data and delete it in accordance with the applicable regulations.

To find your hashed User ID, go to **Settings** > **Preferences** in Tanzu Application Platform GUI.

Overview of security and compliance in Tanzu Application Platform

Security is a primary focus for Tanzu Application Platform (commonly known as TAP).

This section describes:

TLS and certificates in Tanzu Application Platform

Use custom CA certificates in Tanzu Application Platform

Use External Secrets Operator (beta) in Tanzu Application Platform

Assess Tanzu Application Platform against the NIST 800-53 Moderate Assessment

Harden Tanzu Application Platform

Overview of TLS and certificates in Tanzu Application Platform

This topic provides you with information about certificate requirements for ingress and egress communication in Tanzu Application Platform (commonly known as TAP).

Ingress communication uses TLS by default. Platform operators can control certificates and how they are used by components. For more information, see Ingress certificates.

For egress communication, you can establish trust for custom CA certificates. For more information, see Custom CA certificates.

Secure Ingress certificates in Tanzu Application Platform

This topic tells you how to secure exposed ingress endpoints with TLS in Tanzu Application Platform (commonly known as TAP).

Tanzu Application Platform exposes ingress endpoints so that:

- Platform operators and application developers can interact with the platform.
- End users can interact with applications running on the platform.

For information about ingress endpoints and their certificates, see Ingress certificates inventory.

To secure these endpoints with TLS, such as `https://`, Tanzu Application Platform has two ways of configuring ingress certificates:

A shared ingress issuer

VMware recommends a shared ingress issuer as the best practice for issuing ingress certificates on Tanzu Application Platform.

The ingress issuer is an on-platform representation of a certificate authority. All participating components get their certificates issued by it. It is designated by the single Tanzu Application Platform configuration value `shared.ingress_issuer`. Unless customized, all components obtain their ingress certificates from this issuer.

By default, the ingress issuer is self-signed.

For more information about prerequisites, default values, and how to bring your own issuer, see Shared ingress issuer.

Component-level configuration

In some situations, depending on prerequisites, the shared ingress issuer is not the right choice. You can override configuration of TLS and certificates per component. A component's ingress and TLS configuration takes precedence over the shared ingress issuer.

For a list of components with ingress and how to customize them, see Inventory.

Tanzu Application Platform also has limited support for wildcard certificates.

Note

The approaches can be mixed, for example, you can use a shared ingress issuer, but override TLS configuration for select components.

Shared Ingress issuer in Tanzu Application Platform

This topic tells you about the Tanzu Application Platform (commonly known as TAP) shared ingress issuer.

The shared ingress issuer is an on-platform representation of a certificate authority. It provides a method to set up TLS for the entire platform. All participating components get their ingress certificates issued by it.

This is the recommended best practice for issuing ingress certificates on Tanzu Application Platform.

The ingress issuer is designated by the single Tanzu Application Platform configuration value `shared.ingress_issuer`. It refers to a `cert-manager.io/v1/ClusterIssuer`.

By default, a self-signed issuer is used. It's called `tap-ingress-selfsigned` and has limitations. For more information, see Limitations of the default, self-signed issuer.

VMware recommends you replace the default self-signed issuer with your own issuer. For more information, see Replacing the default ingress issuer.

Component-level configuration of TLS takes precedence and can be mixed with the ingress issuer. For more information, see Overriding TLS for components.

You can deactivate the ingress issuer. For more information, see Deactivating TLS for ingress.

Prerequisites

To use the Tanzu Applicatin Platforms ingress issuer your *certificate authority* must be representable by a cert-manager `ClusterIssuer`. You need **one** of the following:

- Your own CA certificate
- Your CA is an ACME, Venafi, or Vault-based issuer, for example, *LetsEncrypt*
- Your CA can be represented by an external cert-manager `ClusterIssuer`.

Without one of the above, you cannot use the issuer ingress, but you can still configure TLS for components. For more information, see Ingress certificates inventory.

Default

By default, Tanzu Application Platform installs and uses a self-signed CA as its ingress issuer for all components.

This default ingress issuer is a self-signed `cert-manager.io/v1/ClusterIssuer` and is provided by Tanzu Application Platform's cert-manager package. Its default name is `tap-ingress-selfsigned`.

The default ingress issuer is appropriate for testing and evaluation, but VMware recommends that you replace it with your own issuer.

Important

If `cert-manager.tanzu.vmware.com` is excluded from the installation, then `tap-ingress-selfsigned` is not installed either. In this case, bring your own ingress issuer.

Limitations of the default, self-signed issuer

The default ingress issuer represents a self-signed *certificate authority*. This is not problematic as far as security is concerned, however, it is not included in any trust chain configured.

As a result, nothing trusts the default ingress issuer implicitly, not even Tanzu Application Platform components. While the issued certificates are valid in principal, they are rejected, for example, by your browser. Furthermore, some interactions between components are not functional by default.

Trusting the default, self-signed issuer

You can trust the default ingress issuer by including `tap-ingress-selfsigned`'s certificate in Tanzu Application Platforms's trusted CA certificates and your device's certificate chain.

Caution

This approach is discouraged. Instead, replace the default ingress issuer.

1. Obtain `tap-ingress-selfsigned`'s PEM-encoded certificate

   ```
   kubectl get secret \
   tap-ingress-selfsigned-root-ca \
   --namespace cert-manager \
   --output go-template='{{ index .data "tls.crt" | base64decode }}'
   ```

2. Add the certificate to custom CA certificates by appending it to `shared.ca_cert_data` and applying the Tanzu Application Platforms installation values file.
3. Add the certificate to your device's trust chain. The trust chain will vary depending on your operating system and privileges.

Replacing the default ingress issuer

Tanzu Application Platform's default ingress issuer can be replaced by any other cert-manager-compliant ClusterIssuer.

To replace the default ingress issuer:

Custom CA

Prerequisites

You need your own CAs certificate and private key for this.

Complete the following steps:

1. Create your `ClusterIssuer`

   Create a `Secret` and `ClusterIssuer` which represent your CA on the platform:

   ```
   ---
   apiVersion: v1
   kind: Secret
   type: kubernetes.io/tls
   metadata:
     name: my-company-ca
     namespace: cert-manager
   stringData:
     tls.crt: #! your CA's PEM-encoded certificate
     tls.key: #! your CA's PEM-encoded private key

   ---
   apiVersion: cert-manager.io/v1
   kind: ClusterIssuer
   metadata:
     name: my-company
   spec:
     ca:
       secretName: my-company-ca
   ```

2. Set `shared.ingress_issuer` to the name of your issuer:

   ```
   #! my-tap-values.yaml
   #! ...
   shared:
     ingress_issuer: my-company-ca
   #! ...
   ```

3. Apply the Tanzu Application Platform installation values file.

   After the configuration is applied, components eventually obtain certificates from the new issuer and will serve them.

LetsEncrypt production

Complete the following steps

Prerequisites

- Public CAs, like LetsEncrypt, record signed certificates in publicly-available certificate logs for the purpose of certificate transparency. Ensure that you are OK with this before using LetsEncrypt.
- LetsEncrypt's production API has rate limits.
- LetsEncrypt requires your `shared.ingress_domain` to be accessible from the Internet.
- Depending on your setup, you might need to adjust .spec.acme.solvers
- Replace `.spec.acme.email` with the email which should receive notices for certificates from LetsEncrypt.

Caution ACME HTTP01 challenges can fail under certain conditions. For more information, see ACME challenges.

1. Create a `ClusterIssuer` for Let's Encrypts production API:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-production
spec:
  acme:
    email: certificate-notices@my-company.com
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour
```

2. Set `shared.ingress_issuer` to the name of your issuer:

```
#! my-tap-values.yaml
#! ...
shared:
  ingress_issuer: letsencrypt-production
#! ...
```

3. Apply Tanzu Application Platform installation values file.

   Once the configuration is applied, components eventually obtain certificates from the new issuer and will serve them.

LetsEncrypt staging

Complete the following steps

Prerequisites

- Public CAs - like LetsEncrypt - record signed certificates in publicly-available certificate logs for the purpose of certificate transparency. Ensure that you are OK with this before using LetsEncrypt.
- LetsEncrypt's staging API is not a publicly-trusted CA. You have to add its certificate to your devices trust chain and Tanzu Application Platform's custom CA certificates.
- LetsEncrypt requires your `shared.ingress_domain` to be accessible from the Internet.
- Depending on your setup you might need to adjust .spec.acme.solvers.
- Replace `.spec.acme.email` with the email which should receive notices for certificates from LetsEncrypt.

Caution

ACME HTTP01 challenges can fail under certain conditions. For more information, see ACME challenges.

1. Create a `ClusterIssuer` for Let's Encrypts staging API:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    email: certificate-notices@my-company.com
```

```
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour
```

2. Set `shared.ingress_issuer` to the name of your issuer:

```
#! my-tap-values.yaml
#! ...
shared:
  ingress_issuer: letsencrypt-staging
#! ...
```

3. Apply Tanzu Application Platform installation values file.

   When the configuration is applied, components obtain certificates from the new issuer and serve them.

Other

Complete the following steps:

You can use any other cert-manager-supported `ClusterIssuer` as an ingress issuer for Tanzu Application Platform.

Cert-manager supports a host of in-tree and out-of-tree issuers. Refer to cert-manager's documentation of issuers.

1. Set `shared.ingress_issuer` to the name of your issuer:

```
#! my-tap-values.yaml
#! ...
shared:
  ingress_issuer: my-company-ca
#! ...
```

2. Apply the Tanzu Application Platform installation values file.

   After the configuration is applied, components obtain certificates from the new issuer and serve them.

There are many ways and tools to assert that new certificates are issued and served. It is best to connect to one of the ingress endpoints and inspect the certificate it serves.

The `openssl` command-line utility is available on most operating systems. The following command retrieves the certificate from an ingress endpoint and shows its text representation:

```
# replace tap.example.com with your Tanzu Application Platform installation's ingress
domain
openssl s_client -showcerts -servername tap-gui.tap.example.com -connect tap-gui.tap.e
xample.com:443 <<< Q | openssl x509 -text -noout
```

Alternatively, use a browser to navigate to the ingress endpoint and click the lock icon in the navigation bar to inspect the certificate.

Deactivating TLS for ingress

While VMware does not recommend it, you can deactivate the ingress issuer by setting `shared.ingress_issuer: ""`.

Overriding TLS for components

You can override TLS settings for each component. In your Tanzu Application Platform values file a component's configuration takes precedence over `shared` values. For more information about which components have ingress and how to configure them, see components.

Note

The approaches can be mixed. Use a shared ingress issuer, but override TLS configuration for select components.

Use wildcard certificates in Tanzu Application Platform

This topic tells you about using wildcard certificates in Tanzu Application Platform (commonly known as TAP) for components with a fixed or variable set of ingress endpoints.

You can use wildcard certificates, but Tanzu Application Platform does not offer support.

Wildcard certificates require component-level configuration. For more information about which components support wildcards, see Inventory.

When using wildcard certificates the approach differs between components that have a fixed set of ingress endpoints and those that have a variable set of ingress endpoints:

- Components with a fixed set of ingress endpoints can receive a reference to the wildcard certificate's `Secret` and an ingress domain, for example, Tanzu Application Platform GUI.
- Components with a variable set of ingress endpoints usually offer Kubernetes APIs that create ingress resources. These components allow configuration of domain templating so that wildcard certificates can be used, for example, Cloud Native Runtimes and Application Single Sign-On.

Note

You can use a mixed approach for configuring TLS for components. For example, you can use a shared ingress issuer, but override TLS configuration for select components while using wildcard certificates for some.

Plan Ingress certificates inventory in Tanzu Application Platform

This topic tells you how to plan for TLS certificates in Tanzu Application Platform (commonly known as TAP).

The effective number of ingress endpoints can vary widely, depending on the installation profile, excluded packages, and end-user-facing resources such as `Workload`, and `AuthServer`. As a result, the number of TLS certificates is not fixed but is a function of the platform's configuration and tenancy.

Components are categorized into those which don't have ingress endpoints and those which do. The latter further breaks down into those which have a fixed number of ingress endpoints and those which offer Kubernetes APIs with ingress.

Note

The lowercase ingress refers to any resource which facilitates ingress, for example, core `Ingress` and Contour's `HTTPProxy`.

Use the following table to help with the planning and accounting of TLS certificates. For a full list of components and the profiles supported for each component, see About Tanzu Application Platform components and profiles.

Package name	Ingress purpose	Supports ingress issuer	Supports wildcards	Number of ingress	SANs*
api-portal.tanzu.vmware.com	Serves the API portal	No	Yes	`1`	`api-portal.INGRESS-DOMAIN`
cnrs.tanzu.vmware.com	Instances of Knative's `Service` have ingress	Yes	Yes	`Number of Services`	SANs depend on the component's `domain_template`
learningcenter.tanzu.vmware.com	Instances of TrainingPortal have ingress	No	Yes**	`Number of TrainingPortals`	`TRAINING-PORTAL.learningcenter.INGRESS-DOMAIN`
metadata-store.apps.tanzu.vmware.com	Serves the Supply Chain Security Tools store	Yes	Yes	`1`	`metadata-store.INGRESS-DOMAIN`
spring-cloud-gateway.tanzu.vmware.com	Instances of SpringCloudGateway have ingress	No	Yes	`Number of SpringCloudGateways`	See Using an Ingress Resource in the Spring Cloud Gateway documentation
sso.apps.tanzu.vmware.com	Instances of AuthServer have ingress	Yes	Yes	`Number of AuthServers`	Depend on the component's `domain_template`
tap-gui.tanzu.vmware.com	Serves the platform-internal developer and service portal	Yes	Yes	`1`	`tap-gui.INGRESS-DOMAIN`

*The SANs is configurable for components in the following two ways:

- components that install a single ingress resource in the form of `COMPONENT.DOMAIN-NAME`, for example, TAP GUI
- components that install an ingress resource per API instance that gets templated from a `domain_template` feeding `DOMAIN-NAME`, for example, `cnrs.tanzu.vmware.com` and `sso.apps.tanzu.vmware.com`

** Only supports wildcards

Use custom CA certificates in Tanzu Application Platform

This topic tells you about configuring custom CA certificates in Tanzu Application Platform (commonly known as TAP).

You configure trust for custom CAs. This is helpful if any Tanzu Application Platforms components are connecting to services that serve certificates issued by private certificate authorities.

The `shared.ca_cert_data` installation value can contain a PEM-encoded CA bundle. Each component then trusts the CAs contained in the bundle.

You can also configure trust per component by providing a CA bundle in the component's installation values. The component then trusts those CAs and the CAs configured in `shared.ca_cert_data`. For more information, see components.

For example:

```
#! my-tap-values.yaml

shared:
  ca_cert_data: |
    Corporate CA 1
    -----BEGIN CERTIFICATE-----
    MIIFmDCCA4....
    -----END CERTIFICATE-----
    Corporate CA 2
    -----BEGIN CERTIFICATE-----
    MIIFkzCCA3....
    -----END CERTIFICATE-----
```

Use External Secrets Operator in Tanzu Application Platform (beta)

The External Secrets Operator is a Kubernetes operator that integrates with external secret management systems, for example, Google Secrets Manager and Hashicorp Vault. It reads information from external APIs and automatically injects the values into a Kubernetes secret.

Tanzu Application Platform (commonly known as TAP) uses the External Secrets Operator to simplify Kubernetes secret life cycle management. The `external-secrets` plug-in, which is available in the Tanzu CLI, interacts with the External Secrets Operator API. Users can use this CLI plug-in to create and view External Secrets Operator resources on a Kubernetes cluster.

External Secrets Operator is available in Tanzu Application Platform packages with a Carvel Package named `external-secrets.apps.tanzu.vmware.com`. It is *not* part of any install profile.

Caution

The External Secrets plug-in is in beta and is intended for evaluation and test purposes only. Do not use it in a production environment.

Where to start

To learn more about managing secrets with External Secrets in general, see the official External Secrets Operator documentation. For installing the External Secrets Operator and the CLI plug-in see the following documentation. Additionally, see the example integration of External-Secrets with Hashicorp Vault.

- Installing External Secrets Operator in TAP
- Installing Tanzu CLI
- External-Secrets with Hashicorp Vault

Install External Secrets Operator in Tanzu Application Platform

This topic tells you how to install the External Secrets Operator from the Tanzu Application Platform (commonly known as TAP) package repository.

Important

External Secrets Operator is not included or installed with any Tanzu Application Platform profile.

Prerequisites

Before installing External Secrets Operator:

- Complete all prerequisites to install the Tanzu Application Platform. For more information, see Prerequisites.

Install

To install External Secrets Operator:

1. List version information for the package by running:

   ```
   tanzu package available list external-secrets.apps.tanzu.vmware.com -n tap-inst
   all
   ```

   For example:

   ```
   NAME                                    VERSION      RELEASED-AT
   external-secrets.apps.tanzu.vmware.com  0.6.1+tap.6  2023-03-08 14:00:00 -0500
   EST
   ```

2. Install the package:

   ```
   tanzu package install external-secrets \
    --package external-secrets.apps.tanzu.vmware.com \
    --version VERSION-NUMBER \
    --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package listed in step 1.

   For example:

   ```
   $ tanzu package install external-secrets \
      --package external-secrets.apps.tanzu.vmware.com \
      --version 0.6.1+tap.6 \
      --namespace tap-install

   \ Installing package 'external-secrets.apps.tanzu.vmware.com'
   | Getting package metadata for 'external-secrets.apps.tanzu.vmware.com'
   | Creating service account 'external-secrets-tap-install-sa'
   / Creating cluster admin role 'external-secrets-tap-install-cluster-role'
   | Creating cluster role binding 'external-secrets-tap-install-cluster-rolebindi
   n
   / Creating cluster role binding 'external-secrets-tap-install-cluster-rolebindi
   n
   \ Creating cluster role binding 'external-secrets-tap-install-cluster-rolebindi
   n
   | Creating cluster role binding 'external-secrets-tap-install-cluster-rolebindi
   ng'
   \ Creating package resource
   Waiting for 'PackageInstall' reconciliation for 'external-secrets'
   ```

```
'PackageInstall' resource install status: Reconciling
'PackageInstall' resource install status: ReconcileSucceeded

Added installed package 'external-secrets'
```

3. Verify the package installation by running:

```
tanzu package installed get external-secrets \
--namespace tap-install
```

For example:

```
tanzu package installed get external-secrets -n tap-install

NAME:                    external-secrets
PACKAGE-NAME:            external-secrets.apps.tanzu.vmware.com
PACKAGE-VERSION:         0.6.1+tap.6
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Integrate External Secrets Operator with HashiCorp Vault in Tanzu Application Platform

This topic shows you how to integrate External Secrets Operator with HashiCorp Vault in Tanzu Application Platform.

The operator synchronizes secret data from external APIs to a Kubernetes secret resource. For more information about Kubernetes secret resources, see the Kubernetes documentation.

Important

This example integration is constructed to showcase the features available and must not be considered in a production environment.

Prerequisites

Before proceeding with this example, you must:

- Install External Secrets Operator. For more information, see Install External Secrets Operator.
- Install the Tanzu CLI. The Tanzu CLI includes the plug-in `external-secrets`. For Tanzu CLI installation, see Tanzu CLI.
- Have a running instance of HashiCorp Vault. In this instance, there is a secret defined with the key `eso-demo/reg-cred`.

Set up the integration

To set up the External Secrets Operator integration with HashiCorp Vault:

1. Create a `Secret` with the vault token. For example:

```
VAULT_TOKEN="vault-token-value"

cat <<EOF | kubectl apply -f -
apiVersion: v1
```

```
kind: Secret
metadata:
name: vault-token
stringData:
token: $VAULT_TOKEN
EOF
```

2. Create a `SecretStore` resource referencing the `vault-token` secret. For example:

> **Caution**
>
> When creating the `SecretStore`, ensure that you match the Vault KV secret engine version. This is either `v1` or `v2`. The default is `v2`. For more information, see Vault KV Secrets Engine documentation.

```
VAULT_SERVER="http://my.vault.server:8200"
VAULT_PATH="eso-demo"

cat <<EOF | tanzu external-secrets store create -y -f -
---
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: vault-secret-store
spec:
  provider:
    vault:
      server: $VAULT_SERVER
      path: $VAULT_PATH
      version: v2
      auth:
        tokenSecretRef:
          name: "vault-token" # vault-token created in the previous step
          key: "token"
EOF
```

3. Verify that the status of the `SecretStore` resource is `Valid` by running:

```
tanzu external-secrets store list
```

Example output:

```
NAMESPACE  NAME                PROVIDER         STATUS
default    vault-secret-store  Hashicorp Vault  Valid
```

4. Create an `ExternalSecret` resource that uses the `SecretStore` you just created by running:

```
cat <<EOF | tanzu external-secrets secret create -y -f -
---
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: vault-secret-example
spec:
  refreshInterval: 15m
  secretStoreRef:
    name: vault-secret-store
    kind: SecretStore
  target:
    name: registry-secret
    template:
```

```
      type: kubernetes.io/dockerconfigjson
      data:
        .dockerconfigjson: "{{ .registryCred | toString }}"
    creationPolicy: Owner
  data:
  - secretKey: registryCred
    remoteRef:
      key: $VAULT_PATH/eso-demo
      property: reg-cred
EOF
```

5. Verify that the status of the `ExternalSecret` resource is `Valid` by running:

   ```
   tanzu external-secrets secret list
   ```

   Example output:

   ```
   NAMESPACE  NAME                   SECRET NAME      STORE               REFRESH I
   NTERVAL  STATUS             LAST UPDATED  LAST REFRESH
   default    vault-secret-example  registry-secret  vault-secret-store  15m
   SecretSynced  21s           10m
   ```

6. After the resource has reconciled, a Kubernetes `secret` resource is created. Look for a secret named `registry-secret` created by the referenced `ExternalSecret`. For example:

   ```
   kubectl get secrets registry-secret -o="jsonpath={.data.\.dockerconfigjson}" |
   base64 -D
   {"auths":{"my-registry.example:8200":{"username":"foo","password":"bar4","emai
   l":"foo@bar.example","auth":"Zm9vOmJhcjQ="}}}
   ```

Assess Tanzu Application Platform against the NIST 800-53 Moderate Assessment

This topic provides you with an assessment of Tanzu Application Platform (commonly known as TAP) against the NIST SP 800-53 Revision 4 Moderate baseline controls. This translates to FISMA Moderate and CNSSI 1253 Mod/Mod/Mod for use in US Federal systems accreditation.

The Moderate baseline applies to only technical controls. Organizational policy controls, physical security, media policies, and similar are excluded as they are not applicable to Tanzu Application Platform. These excluded controls are still relevant to the system at large and must be inherited from existing accreditations or otherwise addressed.

The initial iteration of this assessment delineates responsible parties. Incremental updates will add more details about implementation and updates to 800-53 Revision 5.

Name	Title	Responsible Party	Notes
AC-2(1)	Automated System Account Management	Customer	Implemented on customer identity store. The customer must employ automated mechanisms to support the management of information system accounts used to access their Tanzu Application Platform installation.
AC-2(2)	Removal of Temporary / Emergency Accounts	Customer	Implemented on customer identity store. If the customer chooses to use temporary or emergency accounts, they must ensure that the system automatically deactivates or removes the account following an organization-defined time period.

Name	Title	Responsible Party	Notes
AC-2(3)	Deactivate Inactive Accounts	Customer	Implemented on customer identity store. The customer must automatically deactivate inactive accounts used to access their Tanzu Application Platform installation following an organization-defined time period of inactivity.
AC-2(4)	Automated Audit Actions	Customer	Implemented on customer identity store. The customer must automatically audit account creation, modification, enabling, deactivating, and removal actions associated with accounts used to access their Tanzu Application Platform installation and must notify an organization-defined personnel or role.
AC-3	Access Enforcement	Customer	The customer must federate their IdP with Tanzu Application Platform to enforce approved access authorizations to their Tanzu Application Platform installation.
AC-4	Information Flow Enforcement	Customer	The customer is responsible for enforcing approved authorizations for controlling the flow of information between Tanzu Application Platform and interconnected systems, based on organization-defined information flow control policies, for example, a SIEM. Tanzu Application Platform does not restrict intra-service or inter-system communication. Future versions of Tanzu Application Platform will include this feature using service mesh architecture or similar methods.
AC-6	Least Privilege	Shared	The customer is responsible for enforcing least privilege by ensuring Tanzu Application Platform users have the minimum permissions necessary to perform their job function. Tanzu Application Platform is responsible for providing RBAC functionality to enforce least privilege.
AC-6(1)	Authorize Access to Security Functions	Shared	The customer is responsible for explicitly authorizing access to organization-defined security functions and security-relevant information as it relates to their Tanzu Application Platform installation. Tanzu Application Platform is responsible for providing the RBAC functionality necessary to restrict which users can access security functions and security-related information.
AC-6(5)	Privileged Accounts	Shared	The customer must restrict privileged Tanzu Application Platform accounts to organization-defined personnel or roles. Tanzu Application Platform is responsible for providing the RBAC functionality for customers to restrict privileged Tanzu Application Platform accounts to organization-defined personnel or roles.
AC-6(9)	Auditing Use of Privileged Functions	Shared	The customer is responsible for configuring Tanzu Application Platform and underlying Kubernetes to send log streams to their SIEM tool for log analysis to be capable of auditing the execution of privileged functions. Tanzu Application Platform is responsible for generating logs pertaining to the execution of privileged functions that can be ingested by the customer SIEM tool for analysis.
AC-6(10)	Prohibit Non-Privileged Users from Executing Privileged Functions	Tanzu Application Platform	This functionality is inherent to Tanzu Application Platform/Kubernetes RBAC and can't be configured otherwise.
AC-7 AC-7a AC-7b	Unsuccessful Logon Attempts	Customer	Implemented on customer identity provider. The customer is responsible for configuring their IdP to enforce a limit of consecutive invalid logon attempts by a user during an organization-defined time period which locks the user's account for an organization-defined time period, or until released by an administrator.

Name	Title	Responsible Party	Notes
AC-8 AC-8a AC-8a.1 AC-8a.2 AC-8a.3 AC-8a.4 AC-8b AC-8c AC-8c.1 AC-8c.2 AC-8c.3	System Use Notification	Customer	Implemented on customer identity provider. Customer must configure their IdP to display the system use notification banner before login.
AC-11 AC-11a AC-11b	Session Lock	Customer	The customer must configure sessions locks on user workstations used to access their Tanzu Application Platform installation. Tanzu Application Platform does not have a concept of session locks and relies on sessions locks applied by the user's workstation. Tanzu Application Platform provides logout functionality in place of session locking.
AC-11(1)	Pattern-Hiding Displays	Customer	The customer must configure sessions locks on user workstations used to access their Tanzu Application Platform installation. This includes hiding the user's private session with a publicly available image. Tanzu Application Platform does not have a concept of session locks and relies on sessions locks applied by the user's workstation. Tanzu Application Platform provides logout functionality in place of session locking.
AC-12	Session Termination	Shared	Implemented on customer identity provider. The customer is responsible for configuring IdP token TTL and refresh policies that apply to Tanzu Application Platform sessions. Tanzu Application Platform enforces token policies and cannot be configured otherwise.
AC-14 AC-14a	Permitted Actions Without Identification or Authentication	Shared	The customer is responsible for identifying organization-defined user actions that can be performed on the information system without identification or authentication consistent with organizational missions/business functions. For production installations, Tanzu Application Platform GUI must be configured with OIDC authentication and guest access deactivated.
AC-17(1)	Automated Monitoring / Control	Customer	"Remote Access" is defined as outside-the-org endpoints like remote workers over VPN. This is outside the scope of Tanzu Application Platform. The customer is responsible for all aspects regarding "remote access" to Tanzu Application Platform.
AC-17(2)	Protection of Confidentiality / Integrity Using Encryption	Customer	"Remote Access" is defined as outside-the-org endpoints like remote workers over VPN. This is outside the scope of Tanzu Application Platform. The customer is responsible for implementing cryptographic mechanisms to protect the confidentiality and integrity of "remote access" sessions to Tanzu Application Platform.
AC-17(3)	Managed Access Control Points	Customer	"Remote Access" is defined as outside-the-org endpoints like remote workers over VPN. This is outside the scope of Tanzu Application Platform. The customer is responsible for routing all "remote accesses" to Tanzu Application Platform through an organization-defined number of managed network access control points.
AC-19 AC-19 AC-19b	Access Control for Mobile Devices	Customer	The customer is responsible for all aspects regarding mobile devices which grant access to Tanzu Application Platform.
AU-3	Content of Audit Records	Tanzu Application Platform	The Tanzu Application Platform application must be capable of generating audit logs that contain the minimum content required by the customer consuming the application.

Name	Title	Responsible Party	Notes
AU-3(1)	Additional Audit Information	Customer	Implemented on customer SIEM. The customer is responsible for parsing Tanzu Application Platform logs on their SIEM to extract organization-defined extra information.
AU-4	Audit Storage Capacity	Customer	Implemented on customer Kubernetes. Tanzu Application Platform logs are all captured by Kubernetes logging. The customer is responsible for configuring their Kubernetes hosts with record storage capacity to ensure that there is adequate storage of logs generated by Tanzu Application Platform clusters.
AU-5 AU-5a AU-5b	Response to Audit Processing Failures	Customer	Implemented on customer Kubernetes. Tanzu Application Platform audit records are collected and managed by Kubernetes and are out of Tanzu Application Platform scope. The customer is responsible for configuring their Kubernetes hosts to account for audit processing failures and to alert the appropriate personnel responsible to take appropriate action.
AU-7 AU-7a AU-7b	Audit Reduction and Report Generation	Customer	Implemented on customer Kubernetes and SIEM Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for ensuring that Kubernetes ships Tanzu Application Platform audit records to a central SIEM for review and analysis.
AU-7(1)	Automatic Processing	Customer	Implemented on customer Kubernetes and SIEM Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for ensuring that Kubernetes ships Tanzu Application Platform audit records to a central SIEM for review and analysis.
AU-8 AU-8a AU-8b	Time Stamps	Tanzu Application Platform	Tanzu Application Platform components pull their system time from the container OS and the Kubernetes host and cannot be configured otherwise. Tanzu Application Platform components log statements include UTC timestamps and cannot be configured otherwise.
AU-8(1) AU-8(1)(a) AU-8(1)(b)	Synchronization With Authoritative Time Source	Customer	The customer is responsible for configuring authoritative time sources on K8 clusters.
AU-9	Protection of Audit Information	Customer	Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for protecting Kubernetes and Kubernetes logging configurations from unauthorized access, modification, and deletion.
AU-12 AU-12a AU-12b AU-12c	Audit Generation	Shared	Tanzu Application Platform audit records are collected and managed by Kubernetes. The customer is responsible for ensuring that Kubernetes ships Tanzu Application Platform audit records to a central SIEM for review and analysis. Tanzu Application Platform cannot be configured to audit specific information. Tanzu Application Platform logs verbosely and lets the customer filter out what is relevant to them using their SIEM. Tanzu Application Platform logging cannot be deactivated.
CM-7 CM-7a CM-7b	Least Functionality	Shared	The customer is responsible for configuring Tanzu Application Platform to provide only essential capabilities. Tanzu Application Platform is responsible for providing customers with the capability to deactivate non-essential features not required by the customer. The customer must restrict the use of functions, ports, protocols, and services for the Tanzu Application Platform installation. Tanzu Application Platform is responsible for ensuring that functions, ports, protocols, and services are limited to those explicitly required for the application to operate.

Name	Title	Responsible Party	Notes
CM-7(2)	Prevent Program Execution	Tanzu Application Platform	As an extension of CM-7, Least Functionality, this control is a responsibility of Tanzu Application Platform. Tanzu Application Platform only consists of containers with purposeful services with no extra programs running or bloat. This cannot be configured by the customer.
CM-7(4) (b)	Unauthorized Software/Denylisting	Tanzu Application Platform	Tanzu Application Platform service containers do not implement a deny-by-exception policy to prohibit the execution of unauthorized software programs. Tanzu Application Platform service containers are built to provide stripped-down services and do not include extra programs or bloat. Tanzu Application Platform can provide a SBOM to compare against customer organization policies on disallowed software.
IA-2	Identification and Authentication (Organizational Users)	Shared	The customer is responsible for configuring Tanzu Application Platform to use their IdP which is capable of uniquely identifying and authenticating organizational users. Tanzu Application Platform is responsible for providing customers with the capability to integrate their IdP to allow Tanzu Application Platform to uniquely identify organizational users.
IA-2(1)	Network Access to Privileged Accounts	Customer	Implemented on customer identity provider. The customer is responsible for implementing multifactor authentication on their IdP for network access to privileged accounts.
IA-2(2)	Network Access to Non-Privileged Accounts	Customer	Implemented on customer identity provider. The customer is responsible for implementing multifactor authentication on their IdP for network access to non-privileged accounts.
IA-2(3)	Local Access to Privileged Accounts	N/A	Tanzu Application Platform does not use local accounts. All access occurs over a network connection.
IA-2(8)	Network Access to Privileged Accounts - Replay Resistant	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring that all connections to the customer IdP are over TLS 1.2+.
IA-2(11)	Remote Access - Separate Device	Customer	The customer is responsible for all aspects of MFA and MFA devices used to authenticate to their Tanzu Application Platform installation, including using remote access.
IA-2(12)	Acceptance of Piv Credentials	Customer	Implemented on customer identity provider. The customer is responsible for implementing CAC/PIV credentials with their IdP.
IA-3	Device Identification and Authentication	Customer	The customer is responsible for uniquely identifying and authenticating organization-defined specific and/or types of devices before establishing a local, remote, or network connection.
IA-4e	Identifier Management	Customer	Implemented on customer identity provider. The customer is responsible for configuring IdP token TTL and refresh policies that apply to Tanzu Application Platform sessions. Tanzu Application Platform enforces token policies and cannot be configured otherwise.

Name	Title	Responsible Party	Notes
IA-5(1) IA-5(1)(a) IA-5(1)(b) IA-5(1)(c) IA-5(1)(d) IA-5(1)(e) IA-5(1)(f)	Password-Based Authentication	Customer	Implemented on customer identity store. The customer is responsible for all aspects of password-based authentication to their IdP, using their identity store. Tanzu Application Platform does not employ password-based authentication itself.
IA-5(2) IA-5(2)(a) IA-5(2)(b) IA-5(2)(c) IA-5(2)(d)	PKI-Based Authentication	Customer	Implemented on customer identity provider. The customer is responsible for all aspects of PKI-based authentication on the IdP used to access their Tanzu Application Platform installation.
IA-5(11)	Hardware Token-Based Authentication	Customer	The customer is responsible for ensuring hardware token-based authentication employs mechanisms that satisfy organization-defined token quality requirements.
IA-6	Authenticator Feedback	Customer	Implemented on customer identity provider. The customer is responsible for ensuring their IdP obscures feedback of authentication information during the authentication process.
IA-7	Cryptographic Module Authentication	Customer	Implemented on customer identity provider. The customer is responsible for ensuring their IdP implements FIPS 140-2 validated cryptographic modules.
IA-8	Identification and Authentication(Non-Organizational Users)	Customer	Implemented on customer identity provider. The customer is responsible for ensuring that their IdP uniquely identifies and authenticates non-organizational Tanzu Application Platform users, or processes acting on behalf of non-organizational users.
IA-8(1)	Acceptance of Piv Credentials from Other Agencies	Customer	Implemented on customer identity provider. The customer is responsible for configuring their IdP to accept and electronically verify Personal Identity Verification(PIV) credentials from other federal agencies.
IA-8(2)	Acceptance of Third-Party Credentials	Customer	Implemented on customer identity provider. The customer is responsible for configuring their IdP to accept only FICAM-approved third-party credentials.
IA-8(3)	Use of FICAM-Approved Products	Customer	Implemented on customer identity provider. The customer is responsible for employing only FICAM-approved information system components on their IdP to accept third-party credentials.
IA-8(4)	Use of FICAM-Issued Profiles	Customer	Implemented on customer identity provider. The customer is responsible for ensuring their IdP conforms to FICAM-issued profiles.
SC-2	Application Partitioning	Tanzu Application Platform	Tanzu Application Platform does not isolate user and management functionality on separate network interfaces, instances, CPUs, or similar. Tanzu Application Platform relies on different roles and Kubernetes RBAC to keep user and management functionality distinct.
SC-4	Information in Shared Resources	Tanzu Application Platform	Tanzu Application Platform creates dedicated Kubernetes namespaces upon deployment. Kubernetes namespaces prevent unauthorized and unintended information transfer using shared system resources.

Name	Title	Responsible Party	Notes
SC-5	Denial of Service Protection	Customer	The customer is responsible for ensuring that organizational DoS protections at the network layer include the Tanzu Application Platform installation.
SC-7 SC-7a SC-7b SC-7c	Boundary Protection	Customer	The customer is responsible for the configuration and management of boundary protection devices.
SC-7(4)(c)	External Telecommunications Services	Customer	The customer is responsible for external telecommunication services used to establish connections to their Tanzu Application Platform installation.
SC-7(5)	Deny by Default / Allow by Exception	Shared	Tanzu Application Platform does not implement “deny by default” network policies. This might be mitigated by network-level access controls configured by the customer.
SC-7(7)	Prevent Split Tunneling for Remote Devices	Customer	The customer is responsible for all configuration of remote devices used to access Tanzu Application Platform.
SC-8	Transmission Confidentiality and Integrity	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring all communications occur over TLS 1.2+.
SC-8(1)	Cryptographic or Alternate Physical Protection	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring all communications occur over TLS 1.2+.
SC-10	Network Disconnect	Tanzu Application Platform	Tanzu Application Platform tears down TCP connections and deallocates system resources following the expiration of a session token and cannot be configured otherwise.
SC-12	Cryptographic Key Establishment and Management	Tanzu Application Platform	Tanzu Application Platform is responsible for providing customers with the ability to manage trust stores.
SC-13	Cryptographic Protection	Tanzu Application Platform	Tanzu Application Platform is responsible for implementing FIPS 140 validated cryptographic modules and providing the customer with a means to enable “FIPS Mode”.
SC-21	Secure Name / Address Resolution Service (Recursive or Caching Resolver)	Customer	Tanzu Application Platform inherits the DNSSEC capabilities of the organization resolvers it is configured to use. The customer is responsible for configuring the Tanzu Application Platform and Kubernetes infrastructure to use DNSSEC-capable resolvers.
SC-23	Session Authenticity	Tanzu Application Platform	Tanzu Application Platform is responsible for ensuring all communications occur over TLS 1.2+.

Name	Title	Responsible Party	Notes
SC-28	Protection of Information at Rest	Customer	Tanzu Application Platform does not natively provide encryption for data at rest, but instead relies on the underlying Kubernetes persistent volumes for appropriate cryptographic protections. The customer is responsible for deploying Tanzu Application Platform to Kubernetes with persistent volumes for appropriate cryptographic protections.
SC-39	Process Isolation	Tanzu Application Platform	Tanzu Application Platform container OS enforces the use of separate execution domains for each executing process and cannot be configured otherwise. The underlying Kubernetes host isolates each container from the other.
SI-2c	Flaw Remediation	Tanzu Application Platform	The customer is responsible for keeping the Tanzu Application Platform installation up to date, to within org-defined standards. Tanzu Application Platform does not automatically update itself.
SI-3(2)	Automatic Updates	N/A	Tanzu Application Platform does not include malicious code protection mechanisms therefore automatic update to such mechanisms does not apply.
SI-7(1)	Integrity Checks	Tanzu Application Platform	Tanzu Application Platform performs a hash check when images are downloaded, and a cryptographic signature validation at runtime. This cannot be configured otherwise.
SI-10	Information Input Validation	Tanzu Application Platform	Tanzu Application Platform is responsible for performing input validation of user-supplied input to Tanzu Application Platform.
SI-11 SI-11a SI-11b	Error Handling	Tanzu Application Platform	Tanzu Application Platform limits error message verbosity but does display errors to users. Given the development/coding nature of Tanzu Application Platform, deployment errors and similar must be raised to the user so they can be corrected.
SI-16	Memory Protection	Tanzu Application Platform	Tanzu Application Platform container OS protects its memory from unauthorized code execution and cannot be configured otherwise. The underlying Kubernetes host also isolates container memory pages.

Harden Tanzu Application Platform

This topic provides you with installation and configuration guidance for Tanzu Application Platform (commonly known as TAP) to comply with the NIST 800-53 Security and Privacy Controls for Information Systems and Organizations.

Objective

This is not a comprehensive security guide, but rather, an abbreviated Tanzu Application Platform readiness outline with considerations for hardening Tanzu Application Platform with 800-53 controls as a guide.

Configuring your Tanzu Application Platform installation to this standard does not guarantee approval given there are multiple organizational requirements and deviations that a platform team may make during installation and configuration.

Scope

The document will focus on the hardening on the Tanzu Application Platform. This platform is deployed to Kubernetes and as such, relies on the Kubernetes platform being hardened in a shared responsibility model with the Tanzu Application Platform. This guide will provide instruction on Kubernetes based hardening configurations that are required for the Tanzu Application Platform, however, it should not be viewed as a hardening guide for Kubernetes as well.

For hardening Kubernetes, refer to Kubernetes specific hardening guides such as:

- NSA/CISA Kubernetes Hardening Guide: Published in Aug 2022, this is a prescriptive document that covers many areas related to Kubernetes security.
- NIST Kubernetes STIG Checklist: Published in April 2021, provides a prescriptive a list of technical requirements for securing a basic Kubernetes platform.
- CIS Kubernetes Benchmark: Widely used as a secure configuration guide, last updated in June 2021.

Identity and Access Management

In order to provide an audit trail of what a user does in a system, it is important to configure the Tanzu Application Platform so that the identity for a given user is known. When installing and configuring the Tanzu Application Platform, there are several areas where user identity configuration should be considered. Currently the Tanzu Application Platform has three different areas where users have identities.

1. Tanzu Application Platform GUI
2. Tanzu Application Platform GUI Authentication to Remote Clusters
3. The Kubernetes cluster that the Tanzu Application Platform components are installed on

It is recommended to use the same identity provider for each of these components so that a common identity is shared across the entire Tanzu Application Platform. To facilitate this, components are able to use common OIDC providers. Below is the configuration for each component:

Tanzu Application Platform GUI

The Tanzu Application Platform GUI is based on the Backstage open source project and has a variety of OIDC providers that you are able to configure as an identity provider.

In order to configure authentication for the Tanzu Application Platform GUI, VMware suggests the following:

1. Enable user authentication using one of the supported providers. Note that due to the limitations with the backstage authentication implementation, simply having authentication does not guarantee full end-to-end security as Backstage doesn't currently support per-API authentication. VMware recommends implementing additional security either via an inbound proxy or via networking (firewall / VPN).
2. It is recommended to disable guest access via the tap_gui section in the tap-values.yaml.

```
tap_gui:
 app_config:
   auth:
     allowGuestAccess: false
```

Tanzu Application Platform GUI to Remote Kubernetes Cluster Authentication

Several plugins within the Tanzu Application Platform GUI, such as the Runtime Resource Viewer, Supply Chain Visualization, and Security Analyst GUI require authentication to remote Kubernetes clusters to query Kubernetes resources.

To do so, the plugins must authenticate to the Kubernetes API on remote clusters. This authentication can be configured in two ways: a shared Kubernetes service account that all users will use to authenticate to remote clusters, and by setting up an authentication provider for the

remote cluster. As best security practice, VMware recommends setting up a remote authentication provider for the Kubernetes cluster.

For more information, see Update Tanzu Application Platform GUI to view resources on multiple clusters.

As best practice, the users on the Kubernetes clusters that are used for remote authentication should be assigned to Kubernetes roles that limit access in a least privilege model. More information about Kubernetes roles provider out of box can be found in the next section.

Kubernetes Cluster Authentication and Authorization

Although not a Tanzu Application Platform configuration, VMware recommends enabling authentication to the Kubernetes clusters where the Tanzu Application Platform components are installed, using the same identity provider that other components are using.

While there are many options on how to enable OIDC providers for authentication with the Kubernetes API, VMware supports the Pinniped project and has documented the process of setting it up as part of the Tanzu Application Platform documentation.

By configuring this to use the same identify provider as the Tanzu Application Platform GUI, users can have a common identity across the Kubernetes clusters and the Tanzu Application Platform GUI. Because the Tanzu CLI is making Kubernetes API calls, this configuration will also be enabled for the Tanzu CLI.

Using Pinniped will provide authentication for Kubernetes clusters but still requires the users to be bound to Kubernetes roles. To provide a starting point, the Tanzu Application Platform provides six Kubernetes Roles as part of the installation that users can be bound to. For more information around the roles used for authorization, see Default roles for Tanzu Application Platform.

Cryptographic Protections

Encryption of data is leveraged to prevent unauthorized access to data. With the Tanzu Application Platform, this protection focuses on the two primary states of data that should be encrypted:

1. Encryption of Data in Transit
2. Encryption of Data at Rest

Encryption of Data in Transit

Internal TLS Configuration

Communication between services that originate and terminate within the cluster is referred to as internal communication. Tanzu Application Platform is in the process of enabling TLS on internal communication for components.

If you require encrypted internal communication, there are three remediating options:

1. Enable Tanzu Service Mesh, which provides mutual TLS between components. For more information, see Set up Tanzu Service Mesh.
2. Configure Kubernetes to encrypt all communication with a Container Networking Interface (CNI) that supports traffic encryption, for example, Antrea.
3. Use the underlying network infrastructure running Kubernetes which has encryption on all network traffic.

External TLS Configuration

Based upon OSS doc: https://projectcontour.io/docs/v1.22.1/configuration/#tls-configuration

TLS enables encryption of communication from end-users to the cluster. Since Contour is the edge gateway for all the traffic ingressing into the cluster, it is an easy spot to set up TLS and ensure that all communications between users and the cluster are encrypted.

It also allows cluster owners to satisfy compliance requirements like NIST 800-53 Control SC-8 where it is required to protect the confidentiality of transmitted information.

Moreover, it may be required that certain cipher suites and/or TLS versions are used when encrypting communications.NIST 800-52r2 requires all government-only applications shall use TLS 1.2 and should be configured to use TLS 1.3 as well.

Configuring TLS for Contour

In order to configure Contour to use TLS according to the NIST 800-52r2 requirements you need to create a new section in your `tap-values.yaml` file like:

```
...
contour:
  * existing stuff, probably already there if you're following tap docs
  envoy:
    service:
      type: LoadBalancer # This is set by default, but can be overridden by setting a
different value.
  * new stuff
  contour:
    configFileContents:
      tls:
        minimum-protocol-version: "1.2"
        cipher-suites:
        - '[ECDHE-ECDSA-AES128-GCM-SHA256|ECDHE-ECDSA-CHACHA20-POLY1305]'
        - '[ECDHE-RSA-AES128-GCM-SHA256|ECDHE-RSA-CHACHA20-POLY1305]'
        - 'ECDHE-ECDSA-AES256-GCM-SHA384'
        - 'ECDHE-RSA-AES256-GCM-SHA384'
```

After adding this section, apply the tap-values file and that will change the configuration of TLS to match the requirements.

For more settings in the Contour component, you can reference the open source documentation.

Ingress Certificates

For information about to configure TLS for a Tanzu Application Platform installation's ingress endpoints, see Ingress certificates.

Encryption of Data At Rest

All data should be encrypted at rest. The Tanzu Application Platform runs on Kubernetes and verifies the default storage class configured on the Kubernetes cluster. If you require Encryption of Data at Rest (DARE), you must provide a Persistent Volume Provisioner that supports encryption to the Kubernetes infrastructure.

- Persistent Volume claim encryption
- Data at rest should be encrypted.

Ports and Protocols

Ports are used in TCP and UDP protocols for identification of applications. While some applications use well-known port numbers, such as 80 for HTTP, or 443 for HTTPS, some applications use

dynamic ports. Open port refers to a port on which a system is accepting communication. An open port does not immediately mean a security issue, but it's important to understand that it can provide a pathway for attackers to the application listening on that port. To help with understanding the traffic flows in the Tanzu Application Platform, a list of Tanzu Application Platform ports and protocols is available to existing and future customers upon request.

See the TAP Architecture Overview.

Networking

Ensure that workloads only expose internal-only routes.

All traffic should go through Contour and LoadBalancer without utilizing NodePort services.

Tanzu Application Platform is supported by Tanzu Service Mesh.

You must configure proper affinity rules on Knative deployed services. For more information, see Install Tanzu Application Platform in an air-gapped environment.

Key Management

Key management is the foundation of all data security. Data is encrypted and decrypted via the use of encryption keys or secrets that must be safely stored to prevent the loss or compromise of infrastructure, systems, and applications. Tanzu Application Platform values are secrets and must be protected to ensure the security and integrity of the platform.

- Tanzu Application Platform stores all sensitive values as Kubernetes Secrets
- Encryption of secrets at rest are Kubernetes Distribution Dependent.
- If customers desire to store secrets in a Secret Management service (e.g. Hashicorp Vault, Google Secrets Manager, Amazon Secrets Manager, Microsoft Azure Key Vault) they can make use of the External Secrets Operator to automate the lifecycle management (ALPHA).
- 800-53 Section AC-23 related to safeguarding of sensitive information from exploitation, for example, Tanzu Application Platform values.

Logging

Log files provide an audit trail necessary to monitor activity within infrastructure, identify policy violations, unusual activity, and highlight security incidents. It is vital that logs are captured and retained according to the policies set forth by the organization's security team or governing body. Tanzu Application Platform components run as pods on the Kubernetes infrastructure and all components output to standard out, captured as part of the pod logs.

All Tanzu Application Platform components follow Kubernetes Logging best practices. Log aggregation should be implemented following the best practices of the organization log retention process.

- 800-53 Section AU-4 Audit Log Storage Capacity

Deployment Architecture

Tanzu Application Platform provides a reference architecture that depicts separate components based on function. VMware recommends multiple Kubernetes clusters for the iterate, build, view, and run functions. This separation enables Kubernetes administrators to manage each function independently and therefore, protect the availability and performance of the platform during high usage periods, for example, building or scanning.

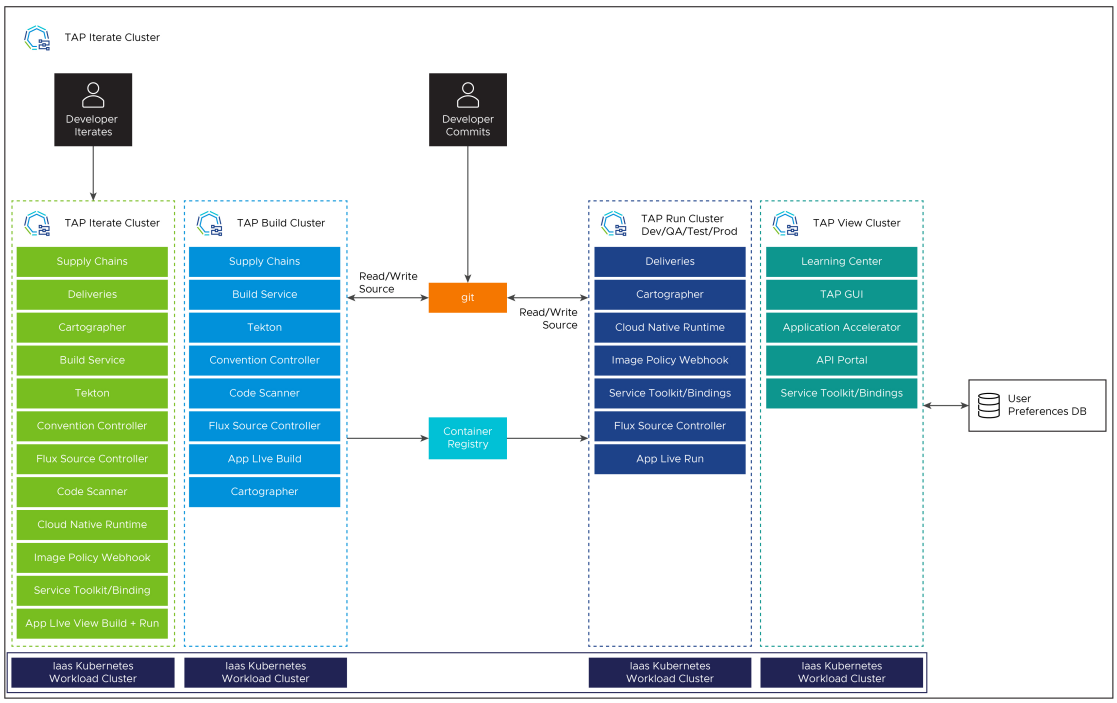
Overview of multicluster Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) in various topologies to reflect your existing landscape. VMware has tested and recommends a multicluster topology for production use. Because flexibility and choice are core to Tanzu Application Platform's design, none of the implementation recommendations are set in stone.

The multicluster topology uses the profile capabilities supported by Tanzu Application Platform. Each cluster adopts one of following multicluster-aligned profiles:

- **Iterate:** Intended for inner-loop iterative application development.
- **Build:** Transforms source revisions to workload revisions; specifically, hosting workloads and supply chains.
- **Run:** Transforms workload revisions to running pods; specifically, hosting deliveries and deliverables.
- **View:** For applications related to centralized developer experiences; specifically, Tanzu Application Platform GUI and metadata store.

The following diagram illustrates this topology.



Next steps

To get started with installing a multicluster topology, see Install multicluster Tanzu Application Platform profiles.

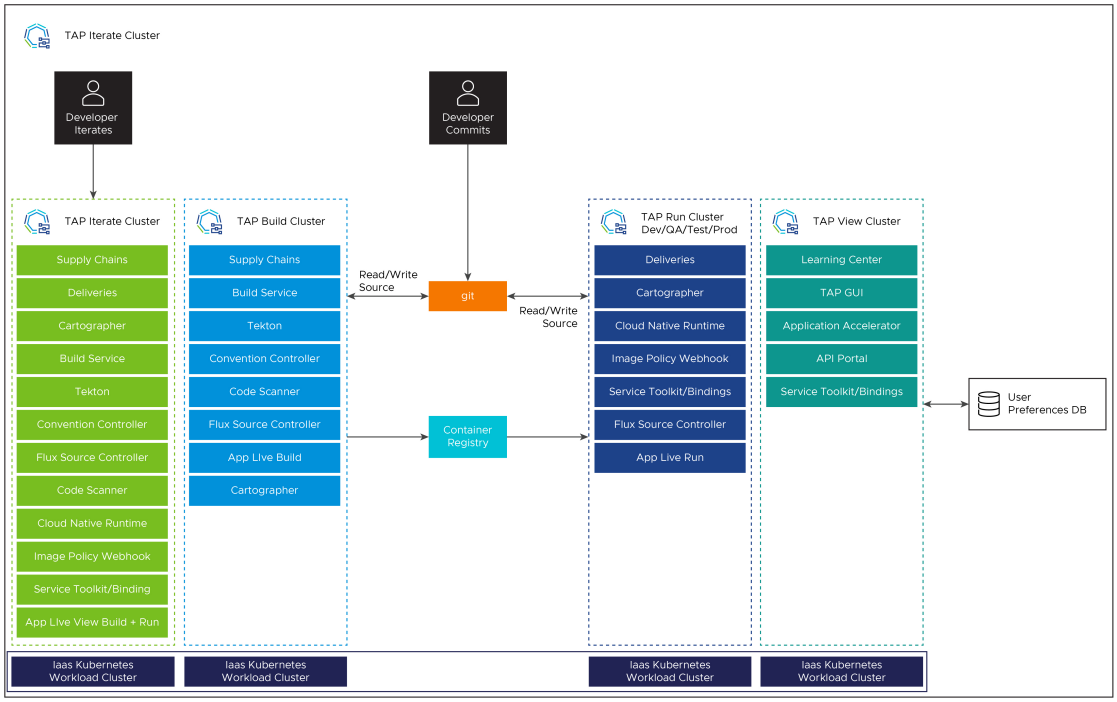
Overview of multicluster Tanzu Application Platform

You can install Tanzu Application Platform (commonly known as TAP) in various topologies to reflect your existing landscape. VMware has tested and recommends a multicluster topology for production use. Because flexibility and choice are core to Tanzu Application Platform's design, none of the implementation recommendations are set in stone.

The multicluster topology uses the profile capabilities supported by Tanzu Application Platform. Each cluster adopts one of following multicluster-aligned profiles:

- **Iterate:** Intended for inner-loop iterative application development.
- **Build:** Transforms source revisions to workload revisions; specifically, hosting workloads and supply chains.
- **Run:** Transforms workload revisions to running pods; specifically, hosting deliveries and deliverables.
- **View:** For applications related to centralized developer experiences; specifically, Tanzu Application Platform GUI and metadata store.

The following diagram illustrates this topology.



Next steps

To get started with installing a multicluster topology, see Install multicluster Tanzu Application Platform profiles.

Install multicluster Tanzu Application Platform profiles

This topic tells you how to install a multicluster topology for your Tanzu Application Platform (commonly known as TAP).

Prerequisites

Before installing multicluster Tanzu Application Platform profiles, you must meet the following prerequisites:

- All clusters must satisfy all the requirements to install Tanzu Application Platform. See Prerequisites.
- Accept Tanzu Application Platform EULA and install Tanzu CLI with any required plug-ins.
- Install Tanzu Cluster Essentials on all clusters. For more information, see Deploy Cluster Essentials.

Multicluster Installation Order of Operations

The installation order is flexible given the ability to update the installation with a modified values file using the `tanzu package installed update` command. The following is an example of the order of operations to be used:

1. Install View profile cluster.
2. Install Build profile cluster.
3. Install Run profile cluster.
4. Install Iterate profile cluster.
5. Add Build, Run and Iterate clusters to Tanzu Application Platform GUI.
6. Update the View cluster's installation values file with the previous information and run the following command to pass the updated config values to Tanzu Application Platform GUI:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION --val
   ues-file tap-values.yaml -n tap-install
   ```

   Where `TAP-VERSION` is the Tanzu Application Platform version you've installed.

Install View cluster

Install the View profile cluster first, because some components must exist before installing the Run clusters. For example, the Application Live View back end must be present before installing the Run clusters. For more information about profiles, see About Tanzu Application Platform package profiles.

To install the View cluster:

1. Follow the steps described in Installing the Tanzu Application Platform package and profiles by using a reduced values file as shown in View profile.
2. Verify that you can access Tanzu Application Platform GUI by using the ingress that you set up. The address must follow this format: `https://tap-gui.INGRESS-DOMAIN`, where `INGRESS-DOMAIN` is the DNS domain you set in `shared.ingress_domain` which points to the shared Contour installation in the `tanzu-system-ingress` namespace with the service `envoy`.
3. Deploy Supply Chain Security Tools (SCST) - Store. See Multicluster setup for more information.

Install Build clusters

To install the Build profile cluster, follow the steps described in Installing the Tanzu Application Platform package and profiles by using a reduced values file as shown in Build profile.

Install Run clusters

To install the Run profile cluster:

1. Follow the steps described in Install the Tanzu Application Platform package and profiles by using a reduced values file as shown in Run profile.
2. To use Application Live View, set the `INGRESS-DOMAIN` for `appliveview_connector` to match the value you set on the View profile for the `appliveview` in the values file.

> **Note**
>
> The default configuration of `shared.ingress_domain` points to the local Run cluster, rather than the View cluster, as a result, `shared.ingress_domain` must be set explicitly.

Install Iterate clusters

To install the Iterate profile cluster, follow the steps described in Install the Tanzu Application Platform package and profiles by using a reduced values file as shown in Iterate profile.

Add Build, Run and Iterate clusters to Tanzu Application Platform GUI

After installing the Build, Run and Iterate clusters, follow the steps in View resources on multiple clusters in Tanzu Application Platform GUI to:

1. Create the `Service Accounts` that Tanzu Application Platform GUI uses to read objects from the clusters.
2. Add a remote cluster.

These steps create the necessary RBAC elements allowing you to pull the URL and token from the Build, Run and Iterate clusters that allows them come back and add to the View cluster's values file.

You must add the Build, Run and Iterate clusters to the View cluster for all plug-ins to function as expected.

Next steps

After setting up the four profiles, you're ready to run a workload by using the supply chain. See Get started with multicluster Tanzu Application Platform.

Get started with multicluster Tanzu Application Platform

This topic tells you how to validate the implementation of a multicluster topology by taking a sample workload and passing it by using the supply chains on the Build and Run clusters.

Use this topic to build an application on the Build profile clusters and run the application on the Run profile clusters.

You can view the workload and associated objects from Tanzu Application Platform GUI (commonly known as TAP GUI) interface on the View profile cluster.

You can take various approaches to configuring the supply chain in this topology, but the following procedures validate the most basic capabilities.

Prerequisites

Before implementing a multicluster topology, complete the following:

1. Complete all installation steps for the four profiles: Build, Run, View and Iterate.
2. For the sample workload, VMware uses the same Application Accelerator - Tanzu Java Web App in the non-multicluster Get Started guide. You can download this accelerator to your own Git infrastructure of choice. You might need to configure additional permissions. Alternatively, you can also use the application-accelerator-samples GitHub repository.
3. The two supply chains are `ootb-supply-chain-basic` on the Build/Iterate profile and `ootb-delivery-basic` on the Run profile. For the Build/Iterate and Run profiled clusters, perform the steps described in Setup Developer Namespace. This guide assumes that you use the `default` namespace.
4. To set the value of `DEVELOPER_NAMESPACE` to the namespace you setup in the previous step, run:

   ```
   export DEVELOPER_NAMESPACE=YOUR-DEVELOPER-NAMESPACE
   ```

   Where:

   - `YOUR-DEVELOPER-NAMESPACE` is the namespace you set up in Set up developer namespaces to use your installed packages. `default` is used in this example.

Start the workload on the Build profile cluster

The Build cluster starts by building the necessary bundle for the workload that is delivered to the Run cluster.

1. Use the Tanzu CLI to start the workload down the first supply chain:

   ```
   tanzu apps workload create tanzu-java-web-app \
   --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
   --sub-path tanzu-java-web-app \
   --git-branch main \
   --type web \
   --label app.kubernetes.io/part-of=tanzu-java-web-app \
   --yes \
   --namespace ${DEVELOPER_NAMESPACE}
   ```

2. To monitor the progress of this process, run:

   ```
   tanzu apps workload tail tanzu-java-web-app --since 10m --timestamp --namespace
   ${DEVELOPER_NAMESPACE}
   ```

3. To exit the monitoring session, press **CTRL** + **C**.
4. Verify that your supply chain has produced the necessary `ConfigMap` containing `Deliverable` content produced by the `Workload`:

   ```
   kubectl get configmap tanzu-java-web-app-deliverable --namespace ${DEVELOPER_NA
   MESPACE} -o go-template='{{.data.deliverable}}'
   ```

   The output resembles the following:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Deliverable
   metadata:
     name: tanzu-java-web-app-deliverable
     labels:
       apis.apps.tanzu.vmware.com/register-api: "true"
       app.kubernetes.io/part-of: tanzu-java-web-app
   ```

```
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/component: deliverable
    app.tanzu.vmware.com/deliverable-type: web
spec:
  params:
  - name: gitops_ssh_secret
    value: ""
  source:
    git:
      url: http://git-server.default.svc.cluster.local/app-namespace/tanzu-java
-web-app
      ref:
        branch: main
```

5. Store the `Deliverable` content, which you can take to the Run profile clusters from the `ConfigMap` by running:

```
kubectl get configmap tanzu-java-web-app-deliverable -n ${DEVELOPER_NAMESPACE}
-o go-template='{{.data.deliverable}}' > deliverable.yaml
```

6. Take this `Deliverable` file to the Run profile clusters by running:

```
kubectl apply -f deliverable.yaml --namespace ${DEVELOPER_NAMESPACE}
```

7. Verify that this `Deliverable` is started and `Ready` by running:

```
kubectl get deliverables --namespace ${DEVELOPER_NAMESPACE}
```

   The output resembles the following:

```
kubectl get deliverables --namespace default
NAME                 SOURCE
DELIVERY         READY   REASON   AGE
tanzu-java-web-app   tapmulticloud.azurecr.io/tap-multi-build-dev/tanzu-java-we
b-app-default-bundle:xxxx-xxxx-xxxx-xxxx-1a7beafd6389   delivery-basic   True
Ready    7m2s
```

8. To test the application, query the URL for the application. Look for the `httpProxy` by running:

```
kubectl get httpproxy --namespace ${DEVELOPER_NAMESPACE}
```

   The output resembles the following:

```
kubectl get httpproxy --namespace default
NAME                                                                FQDN
TLS SECRET   STATUS   STATUS DESCRIPTION
tanzu-java-web-app-contour-a98df54e3629c5ae9c82a395501ee1fdtanz   tanzu-java-we
b-app.default.svc.cluster.local                                 valid    Valid HTTPP
roxy
tanzu-java-web-app-contour-e1d997a9ff9e7dfb6c22087e0ce6fd7ftanz   tanzu-java-we
b-app.default.apps.run.multi.kapplegate.com                     valid    Valid HTTPP
roxy
tanzu-java-web-app-contour-tanzu-java-web-app.default             tanzu-java-we
b-app.default                                                   valid    Valid HTTPP
roxy
tanzu-java-web-app-contour-tanzu-java-web-app.default.svc         tanzu-java-we
b-app.default.svc                                               valid    Valid HTTPP
roxy
```

   Select the URL that corresponds to the domain you specified in your Run cluster's profile and enter it into a browser. Expect to see the message "Greetings from Spring Boot +

Tanzu!”.

9. View the component in Tanzu Application Platform GUI, by following these steps and using the catalog file from the sample accelerator in GitHub.

Install Tanzu Application Platform Build profile

This topic tells you how to install Build profile cluster by using a reduced values file.

Prerequisites

Before installing the Build profile, follow all the steps in Install View cluster.

Example values.yaml

The following is the YAML file sample for the build-profile:

```
profile: build
ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

shared:
  ingress_domain: "INGRESS-DOMAIN"
  kubernetes_distribution: "openshift" # To be passed only for Openshift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME" # To be used by Build Service by appending
"/buildservice" and used by Supply chain by appending "/workloads".
    secret:
      name: "KP-DEFAULT-REPO-SECRET"
      namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----

# The above shared keys can be overridden in the below section.

buildservice:
# Takes the value from the shared section by default, but can be overridden by setting
a different value.
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_secret:
    name: "KP-DEFAULT-REPO-SECRET"
    namespace: "KP-DEFAULT-REPO-SECRET-NAMESPACE"
supply_chain: testing_scanning
ootb_supply_chain_testing_scanning: # Optional if the corresponding shared keys are pr
ovided.
  registry:
    server: "SERVER-NAME"
    repository: "REPO-NAME"
  gitops:
    ssh_secret: "SSH-SECRET-KEY" # (Optional) Defaults to "".
grype:
  namespace: "MY-DEV-NAMESPACE" # (Optional) Defaults to default namespace.
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
  metadataStore:
    url: METADATA-STORE-URL-ON-VIEW-CLUSTER
    caSecret:
        name: store-ca-cert
        importFromNamespace: metadata-store-secrets
    authSecret:
```

```
        name: store-auth-token
        importFromNamespace: metadata-store-secrets
scanning:
  metadataStore:
    url: "" # Configuration is moved, so set this string to empty.
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.
```

Where:

- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `KP-DEFAULT-REPO` is a writable repository in your registry. The Tanzu Build Service dependencies are written to this location. Examples:
  - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`
  - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`
  - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`
  - For Google Cloud Registry, use the contents of the service account JSON file.
- `KP-DEFAULT-REPO-SECRET` is the secret with user credentials that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
  - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
  - You must create the secret before the installation. For example, you can use the `registry-credentials` secret created earlier.
- `KP-DEFAULT-REPO-SECRET-NAMESPACE` is the namespace where `KP-DEFAULT-REPO-SECRET` is created.
- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `server: "my-harbor.io"`.
  - Docker Hub has the form `server: "index.docker.io"`.
  - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `repository: "my-project/supply-chain"`.
  - Docker Hub has the form `repository: "my-dockerhub-user"`.
  - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. See Git authentication for more information.
- `METADATA-STORE-URL-ON-VIEW-CLUSTER` is the URL of the Supply Chain Security Tools (SCST) - Store deployed on the View cluster. For example, `https://metadata-store.example.com`. For information about `caSecret` and `store-auth-token`, see Multicluster setup.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Scan deploys the `ScanTemplates` there. This allows the scanning feature to run in this namespace.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the Secret that contains the credentials to pull an image from the registry for scanning.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports. See the Tanzu Kubernetes Grid documentation for more information about identifying the Entitlement Account Number.

When you install Tanzu Application Platform, it is bootstrapped with the `lite` set of dependencies, including buildpacks and stacks, for application builds. For more information about buildpacks, see the VMware Tanzu Buildpacks Documentation. You can find the buildpack and stack artifacts installed with Tanzu Application Platform on Tanzu Network. You can update the dependencies by upgrading Tanzu Application Platform to the latest patch.

See Multicluster setup for more information about the value settings of `grype.metadataStore`.

You must set the `scanning.metadatastore.url` to an empty string if you're installing Grype Scanner v1.2.0 and later or Snyk Scanner to deactivate the embedded SCST - Store integration.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

Install Tanzu Application Platform Run profile

This topic tells you how to install Run profile cluster by using a reduced values file.

The following is the YAML file sample for the run-profile:

```
profile: run
ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

shared:
  ingress_domain: INGRESS-DOMAIN
  kubernetes_distribution: "openshift" # To be passed only for Openshift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----
supply_chain: basic

contour:
  envoy:
    service:
      type: LoadBalancer # NodePort can be used if your Kubernetes cluster doesn't sup
port LoadBalancing.

appliveview_connector:
  backend:
    sslDeactivated: TRUE-OR-FALSE-VALUE
    ingressEnabled: true
    host: appliveview.VIEW-CLUSTER-INGRESS-DOMAIN

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's external IP address.
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `VIEW-CLUSTER-INGRESS-DOMAIN` is the subdomain you setup on the View profile cluster. This matches the value key `appliveview.ingressDomain` or `shared.ingress_domain` on the view cluster. Include the default host name `appliveview.` ahead of the domain.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports. See the Tanzu Kubernetes Grid documentation for more information about identifying the Entitlement Account Number.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

If you set `shared.ingress_domain` in run profile, the `appliveview_connector.backend.host` is automatically configured as `host: appliveview.INGRESS-DOMAIN`. To override the shared ingress for Application Live View to connect to the view cluster, set the `appliveview_connector.backend.host` key to `appliveview.VIEW-CLUSTER-INGRESS-DOMAIN`.

Install Tanzu Application Platform View profile

This topic tells you how to install View profile cluster by using a reduced values file.

The following is the YAML file sample for the view-profile:

```
profile: view
ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

shared:
  ingress_domain: "INGRESS-DOMAIN"
  kubernetes_distribution: "openshift" # To be passed only for Openshift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  ca_cert_data: | # To be passed if using custom certificates.
    -----BEGIN CERTIFICATE-----
    MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
    -----END CERTIFICATE-----

contour:
  envoy:
    service:
      type: LoadBalancer # NodePort can be used if your Kubernetes cluster doesn't sup
port LoadBalancing.

tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: https://GIT-CATALOG-URL/catalog-info.yaml
    kubernetes:
      serviceLocatorMethod:
        type: 'multiTenant'
```

```
      clusterLocatorMethods:
        - type: 'config'
          clusters:
            - url: CLUSTER-URL
              name: CLUSTER-NAME # Build profile cluster can go here.
              authProvider: serviceAccount
              serviceAccountToken: CLUSTER-TOKEN
              skipTLSVerify: TRUE-OR-FALSE-VALUE
            - url: CLUSTER-URL
              name: CLUSTER-NAME # Run profile cluster can go here.
              authProvider: serviceAccount
              serviceAccountToken: CLUSTER-TOKEN
              skipTLSVerify: TRUE-OR-FALSE-VALUE

appliveview:
  ingressEnabled: true

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.
```

Where:

- `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's external IP address.
- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. You can download either a blank or populated catalog file from the Tanzu Application Platform product page. Otherwise, use a Backstage-compliant catalog you've already built and posted on the Git infrastructure in the Integration section.
- `CLUSTER-URL`, `CLUSTER-NAME` and `CLUSTER-TOKEN` are described in the View resources on multiple clusters in Tanzu Application Platform GUI. Observe the order of operations laid out in the previous steps.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports. See the Tanzu Kubernetes Grid documentation for more information about identifying the Entitlement Account Number.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

The `appliveview.ingressEnabled` key is set to `false` by default. In a multicluster setup, `ingressEnabled` key must be set to `true`. If the `shared.ingress_domain` key is set, the Application Live View back end is automatically exposed through the shared ingress.

Install Tanzu Application Platform Iterate profile

This topic tells you how to install Iterate profile cluster by using a reduced values file.

The following is the YAML file sample for the iterate-profile:

```
profile: iterate
```

```
shared:
  ingress_domain: "INGRESS-DOMAIN"
  kubernetes_distribution: "openshift" # To be passed only for OpenShift. Defaults to
"".
  kubernetes_version: "K8S-VERSION"
  image_registry:
    project_path: "SERVER-NAME/REPO-NAME" # To be used by Build Service by appending
"/buildservice" and used by Supply chain by appending "/workloads"
    username: "KP-DEFAULT-REPO-USERNAME"
    password: "KP-DEFAULT-REPO-PASSWORD"
  ca_cert_data: | # To be passed if using custom certificates
  -----BEGIN CERTIFICATE-----
  MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
  -----END CERTIFICATE-----

ceip_policy_disclosed: FALSE-OR-TRUE-VALUE # Installation fails if this is not set to
true. Not a string.

# The above shared keys may be overridden in the below section.

buildservice: # Optional if the corresponding shared keys are provided.
  kp_default_repository: "KP-DEFAULT-REPO"
  kp_default_repository_username: "KP-DEFAULT-REPO-USERNAME"
  kp_default_repository_password: "KP-DEFAULT-REPO-PASSWORD"

supply_chain: basic
ootb_supply_chain_basic: # Optional if the shared above mentioned shared keys are prov
ided.
  registry:
    server: "SERVER-NAME"
    repository: "REPO-NAME"
  gitops:
    ssh_secret: "SSH-SECRET-KEY" # (Optional) Defaults to "".

image_policy_webhook:
  allow_unmatched_tags: true

contour:
  envoy:
    service:
      type: LoadBalancer # (Optional) Defaults to LoadBalancer.

cnrs:
  domain_name: "TAP-ITERATE-CNRS-DOMAIN" # Optional if the shared.ingress_domain is pr
ovided.

appliveview_connector:
  backend:
    sslDeactivated: TRUE-OR-FALSE-VALUE
    ingressEnabled: true
    host: appliveview.VIEW-CLUSTER-INGRESS-DOMAIN

tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER" # (Option
al) Identify data for creating Tanzu Application Platform usage reports.
```

Where:

- `K8S-VERSION` is the Kubernetes version used by your OpenShift cluster. It must be in the form of `1.23.x` or `1.24.x`, where `x` stands for the patch version. Examples:
  - Red Hat OpenShift Container Platform v4.10 uses the Kubernetes version `1.23.3`.
  - Red Hat OpenShift Container Platform v4.11 uses the Kubernetes version `1.24.1`.
- `KP-DEFAULT-REPO` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:

    - Harbor has the form `kp_default_repository: "my-harbor.io/my-project/build-service"`.
    - Docker Hub has the form `kp_default_repository: "my-dockerhub-user/build-service"` or `kp_default_repository: "index.docker.io/my-user/build-service"`.
    - Google Cloud Registry has the form `kp_default_repository: "gcr.io/my-project/build-service"`.
- `KP-DEFAULT-REPO-USERNAME` is the user name that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential.
    - For Google Cloud Registry, use `kp_default_repository_username: _json_key`.
- `KP-DEFAULT-REPO-PASSWORD` is the password for the user that can write to `KP-DEFAULT-REPO`. You can `docker push` to this location with this credential. This credential can also be configured by using a Secret reference. For more information, see Install Tanzu Build Service for details.
    - For Google Cloud Registry, use the contents of the service account JSON file.
- `SERVER-NAME` is the host name of the registry server. Examples:
    - Harbor has the form `server: "my-harbor.io"`.
    - Docker Hub has the form `server: "index.docker.io"`.
    - Google Cloud Registry has the form `server: "gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
    - Harbor has the form `repository: "my-project/supply-chain"`.
    - Docker Hub has the form `repository: "my-dockerhub-user"`.
    - Google Cloud Registry has the form `repository: "my-project/supply-chain"`.
- `SSH-SECRET-KEY` is the SSH secret key in the developer namespace for the supply chain to fetch source code from and push configuration to. See Git authentication for more information.
- `TAP-ITERATE-CNRS-DOMAIN` is the iterate cluster CNRS domain.
- `VIEW-CLUSTER-INGRESS-DOMAIN` is the subdomain you setup on the View profile cluster. This matches the value key `appliveview.ingressDomain` or `shared.ingress_domain` on the view cluster. Include the default host name `appliveview.` ahead of the domain.
- `CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER` (optional) refers to the Entitlement Account Number (EAN), which is a unique identifier VMware assigns to its customers. Tanzu Application Platform telemetry uses this number to identify data that belongs to a particular customers and prepare usage reports. See the Tanzu Kubernetes Grid documentation for more information about identifying the Entitlement Account Number.

If you use custom CA certificates, you must provide one or more PEM-encoded CA certificates under the `ca_cert_data` key. If you configured `shared.ca_cert_data`, Tanzu Application Platform component packages inherit that value by default.

If you set `shared.ingress_domain` in the iterate profile, the `appliveview_connector.backend.host` is automatically configured as `host: appliveview.INGRESS-DOMAIN`. To override the shared ingress for Application Live View to connect to the view cluster, set the `appliveview_connector.backend.host` key to `appliveview.VIEW-CLUSTER-INGRESS-DOMAIN`.

Get started with Tanzu Application Platform

Welcome to Tanzu Application Platform (commonly known as TAP). The guides in this section provide hands-on instructions for developers and operators to help you get started on Tanzu Application Platform.

Prerequisites

Before you start, verify you have successfully:

- **Installed Tanzu Application Platform**
  See Installing Tanzu Application Platform.
- **Installed Tanzu Application Platform on the target Kubernetes cluster**
  See Installing the Tanzu CLI and Installing the Tanzu Application Platform Package and Profiles.
- **Set the default kubeconfig context to the target Kubernetes cluster**
  See Changing clusters.
- **Installed Out of The Box (OOTB) Supply Chain Basic**
  See Install Out of The Box Supply Chain Basic. If you used the default profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed the Out of The Box (OOTB) Supply Chain Basic.
- **Installed Tekton Pipelines**
  See Install Tekton Pipelines. If you used the default profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tekton Pipelines.
- **Set up a developer namespace to accommodate the developer workload**
  See Set up developer namespaces to use your installed packages.
- **Installed Tanzu Application Platform GUI**
  See Install Tanzu Application Platform GUI. If you used the Full or View profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tanzu Application Platform GUI.
- **Installed the VS Code Tanzu Extension**
  See Install the Visual Studio Code Tanzu Extension for instructions.

When you have completed these prerequisites, you are ready to get started.

Next steps

For developers:

- Create an application accelerator
- Deploy an app on Tanzu Application Platform
- Deploy Spring Cloud Applications to Tanzu Application Platform

For operators:

- Add testing and scanning to your application
- Configure image signing

Get started with Tanzu Application Platform

Welcome to Tanzu Application Platform (commonly known as TAP). The guides in this section provide hands-on instructions for developers and operators to help you get started on Tanzu Application Platform.

Prerequisites

Before you start, verify you have successfully:

- **Installed Tanzu Application Platform**
  See Installing Tanzu Application Platform.
- **Installed Tanzu Application Platform on the target Kubernetes cluster**
  See Installing the Tanzu CLI and Installing the Tanzu Application Platform Package and Profiles.
- **Set the default kubeconfig context to the target Kubernetes cluster**
  See Changing clusters.
- **Installed Out of The Box (OOTB) Supply Chain Basic**
  See Install Out of The Box Supply Chain Basic. If you used the default profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed the Out of The Box (OOTB) Supply Chain Basic.
- **Installed Tekton Pipelines**
  See Install Tekton Pipelines. If you used the default profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tekton Pipelines.
- **Set up a developer namespace to accommodate the developer workload**
  See Set up developer namespaces to use your installed packages.
- **Installed Tanzu Application Platform GUI**
  See Install Tanzu Application Platform GUI. If you used the Full or View profiles provided in Installing the Tanzu Application Platform Package and Profiles, you have already installed Tanzu Application Platform GUI.
- **Installed the VS Code Tanzu Extension**
  See Install the Visual Studio Code Tanzu Extension for instructions.

When you have completed these prerequisites, you are ready to get started.

Next steps

For developers:

- Create an application accelerator
- Deploy an app on Tanzu Application Platform
- Deploy Spring Cloud Applications to Tanzu Application Platform

For operators:

- Add testing and scanning to your application
- Configure image signing

Add testing and scanning to your application

This topic guides you through installing the optional OOTB Supply Chain with Testing and the optional OOTB Supply Chain with Testing and Scanning.

For more information about available supply chains, see Supply chains on Tanzu Application Platform.

What you will do

- Install OOTB Supply Chain with Testing.
- Add a Tekton pipeline to the cluster and update the workload to point to the pipeline and resolve errors.
- Install OOTB Supply Chain with Testing and Scanning.
- Update the workload to point to the Tekton pipeline and resolve errors.
- Query for vulnerabilities and dependencies.

Overview

The default Out of the Box (OOTB) Supply Chain Basic and its dependencies were installed on your cluster during the Tanzu Application Platform install. As demonstrated in this guide, you can add testing and security scanning to your application. When you activate OOTB Supply Chain with Testing, it deactivates OOTB Supply Chain Basic.

The following installations also provide a sample Tekton pipeline that tests your sample application. The pipeline is configurable. Therefore, you can customize the steps to perform either additional testing or other tasks with Tekton Pipelines.

Install OOTB Supply Chain with Testing

To install OOTB Supply Chain with Testing:

1. You can activate the OOTB Supply Chain with Testing by updating your profile to use `testing` rather than `basic` as the selected supply chain for workloads in this cluster. The `tap-values.yaml` is the file used to customize the profile in `Tanzu package install tap --values-file=...`. Update `tap-values.yaml` with the following changes:

   ```
   - supply_chain: basic
   + supply_chain: testing

   - ootb_supply_chain_basic:
   + ootb_supply_chain_testing:
       registry:
         server: "SERVER-NAME"
         repository: "REPO-NAME"
   ```

   Where:

   - `SERVER-NAME` is the name of your server.
   - `REPO-NAME` is the name of the image repository that hosts the container images.

2. Update the installed profile by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION-NUMBER --values-file tap-values.yaml -n tap-install
   ```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.

Tekton pipeline config example

In this section, a Tekton pipeline is added to the cluster. In the next section, the workload is updated to point to the pipeline and resolve any current errors.

Note

Developers can perform this step because they know how their application must be tested. The operator can also add the Tekton pipeline to a cluster before the developer gets access.

To add the Tekton pipeline to the cluster, apply the following YAML to the cluster:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  namespace: DEVELOPER-NAMESPACE
  labels:
    apps.tanzu.vmware.com/pipeline: test     # (!) required
spec:
  params:
    - name: source-url                       # (!) required
    - name: source-revision                  # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`

              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

Where `DEVELOPER-NAMESPACE` is the name of your developer namespace.

The preceding YAML puts a Tekton pipeline in the developer namespace you specify. It defines the Tekton pipeline with a single step. The step contained in the `steps` pulls the code from the repository indicated in the developers `workload` and runs the tests within the repository. The steps of the Tekton pipeline are configurable and allow the developer to add additional items needed to test their code.

There are many steps in the supply chain. In this case, the next step is an image build. Any additional steps the developer adds to the Tekton pipeline to test their code are independent of the image being built and of any subsequent steps executed in the supply chain. This independence gives the developer freedom to focus on testing their code.

The `params` are templated by Supply Chain Choreographer. Additionally, Tekton pipelines require a Tekton `pipelineRun` to execute on the cluster. Supply Chain Choreographer handles creating the

`pipelineRun` dynamically each time that step of the supply chain requires execution.

Workload update

To connect the new supply chain to the workload, the workload must be updated to point at your Tekton pipeline.

1. Update the workload by running the following with the Tanzu CLI:

   ```
   tanzu apps workload apply tanzu-java-web-app \
     --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
     --sub-path tanzu-java-web-app \
     --git-branch main \
     --type web \
     --label apps.tanzu.vmware.com/has-tests=true \
     --label app.kubernetes.io/part-of=tanzu-java-web-app \
     --yes
   ```

2. After accepting the workload creation, monitor the creation of new resources by the workload by running:

   ```
   kubectl get workload,gitrepository,pipelinerun,images.kpack,podintent,app,servi
   ces.serving
   ```

   The result is output similar to the following example that shows the objects created by Supply Chain Choreographer:

   ```
   NAME                                      AGE
   workload.carto.run/tanzu-java-web-app   109s

   NAME                                                              URL
   READY   STATUS                                                                AGE
   gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app   https://github.com/
   vmware-tanzu/application-accelerator-samples   True    Fetched revision: main/8
   72ff44c8866b7805fb2425130edb69a9853bfdf   109s

   NAME                                                SUCCEEDED   REASON      START
   TIME   COMPLETIONTIME
   pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb   True        Succeeded   104s
   77s

   NAME                                LATESTIMAGE
   READY
   image.kpack.io/tanzu-java-web-app   10.188.0.3:5000/foo/tanzu-java-web-app@sha2
   56:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c   True

   NAME                                                                READY   REASON
   AGE
   podintent.conventions.carto.run/tanzu-java-web-app   True             7s

   NAME                                        DESCRIPTION           SINCE-DEPLOY
   AGE
   app.kappctrl.k14s.io/tanzu-java-web-app   Reconcile succeeded   1s
   2s

   NAME                                                URL
   LATESTCREATED              LATESTREADY                READY     REASON
   service.serving.knative.dev/tanzu-java-web-app   http://tanzu-java-web-app.deve
   loper.example.com   tanzu-java-web-app-00001   tanzu-java-web-app-00001   Unkno
   wn   IngressNotConfigured
   ```

Install OOTB Supply Chain with Testing and Scanning

Prerequisites

- Both the Scan Controller and the default Grype scanner must be installed for scanning. Refer to the verify installation steps later in the topic.

  > **Note**
  >
  > When leveraging both Tanzu Build Service and Grype in your Tanzu Application Platform supply chain, you can receive enhanced scanning coverage for the languages and frameworks with check marks in the column "Extended Scanning Coverage using Anchore Grype" on the Language and Framework Support Table.

- Add the necessary configuration to enable CVE scan results in the Tanzu Application Platform GUI. This configuration allows the Supply Chain Choreographer Tanzu Application Platform GUI plug-in to retrieve metadata about project packages and their vulnerabilities.

To install OOTB Supply Chain with Testing and Scanning:

1. Supply Chain Security Tools (SCST) - Scan is installed as part of the Tanzu Application Platform profiles. Verify that both Scan Controller and Grype Scanner are installed by running:

   ```
   tanzu package installed get scanning -n tap-install
   tanzu package installed get grype -n tap-install
   ```

   If the packages are not already installed, follow the steps in Supply Chain Security Tools - Scan to install the required scanning components.

   During installation of the Grype Scanner, sample ScanTemplates are installed into the `default` namespace. If the workload is deployed into another namespace, these sample ScanTemplates must also be present in the other namespace. One way to accomplish this is to install Grype Scanner again and provide the namespace in the values file.

   A ScanPolicy is required and must be in the required namespace. A sample ScanPolicy is provided as follows to block a supply chain when CVEs with critical, high, and unknown ratings are found using `notAllowedSeverities := ["Critical","High","UnknownSeverity"]`. You can also configure the supply chain to use your own custom policies and apply exceptions when you want to ignore certain CVEs. See Out of the Box Supply Chain with Testing and Scanning. To apply the sample ScanPolicy, you can either add the namespace flag to the kubectl command or add the namespace text box to the template by running:

   ```
   kubectl apply -f - -o yaml << EOF
   ---
   apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
   kind: ScanPolicy
   metadata:
     name: scan-policy
     labels:
       'app.kubernetes.io/part-of': 'enable-in-gui'
   spec:
     regoFile: |
       package main

       # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
   ownSeverity"
       notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
       ignoreCves := []
   ```

```
    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
EOF
```

2. (optional) The Tanzu Application Platform profiles install the Supply Chain Security Tools - Store package by default. To persist and query the vulnerability results post-scan, confirm it is installed by running:

   ```
   tanzu package installed get metadata-store -n tap-install
   ```

   If the package is not installed, follow the installation instructions at Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles.

3. Update the profile to use the supply chain with testing and scanning by updating `tap-values.yaml` (the file used to customize the profile in `tanzu package install tap --values-file=...`) with the following changes:

   ```
   - supply_chain: testing
   + supply_chain: testing_scanning

   - ootb_supply_chain_testing:
   + ootb_supply_chain_testing_scanning:
       registry:
         server: "<SERVER-NAME>"
         repository: "<REPO-NAME>"
   ```

4. Update the `tap` package:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION-NUMBER --
   values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.

Workload update

To connect the new supply chain to the workload, update the workload to point to your Tekton pipeline:

1. Update the workload by running the following using the Tanzu CLI:

```
tanzu apps workload apply tanzu-java-web-app \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --git-branch main \
  --type web \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --yes
```

2. After accepting the workload creation, view the new resources that the workload created by running:

```
kubectl get workload,gitrepository,sourcescan,pipelinerun,images.kpack,imagesca
n,podintent,app,services.serving
```

   The following is an example output, which shows the objects that Supply Chain Choreographer created:

```
NAME                                      AGE
workload.carto.run/tanzu-java-web-app   109s

NAME                                                                    URL
READY   STATUS                                                                  AGE
gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app   https://github.com/
vmware-tanzu/application-accelerator-samples   True    Fetched revision: main/8
72ff44c8866b7805fb2425130edb69a9853bfdf   109s

NAME                                                                PHASE       SCAN
NEDREVISION                              SCANNEDREPOSITORY
AGE    CRITICAL   HIGH   MEDIUM   LOW   UNKNOWN   CVETOTAL
sourcescan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app   Completed   1878
50b39b754e425621340787932759a0838795   https://github.com/vmware-tanzu/applicat
ion-accelerator-samples   90s

NAME                                                SUCCEEDED   REASON      START
TIME   COMPLETIONTIME
pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb   True        Succeeded   104s
77s

NAME                                LATESTIMAGE
READY
image.kpack.io/tanzu-java-web-app   10.188.0.3:5000/foo/tanzu-java-web-app@sha2
56:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c   True

NAME                                                                PHASE       SCANN
EDIMAGE
AGE   CRITICAL   HIGH   MEDIUM   LOW   UNKNOWN   CVETOTAL
imagescan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app   Completed   10.18
8.0.3:5000/foo/tanzu-java-web-app@sha256:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b8965
46e005f1efd98fbc4e79b7552c   14s

NAME                                                                  READY   REASON
AGE
podintent.conventions.carto.run/tanzu-java-web-app   True             7s

NAME                                     DESCRIPTION           SINCE-DEPLOY
AGE
```

```
app.kappctrl.k14s.io/tanzu-java-web-app   Reconcile succeeded   1s
2s

NAME                                                URL
LATESTCREATED              LATESTREADY                  READY     REASON
service.serving.knative.dev/tanzu-java-web-app   http://tanzu-java-web-app.deve
loper.example.com   tanzu-java-web-app-00001   tanzu-java-web-app-00001   Unkno
wn   IngressNotConfigured
```

Important

If the source or image scan has a “Failed” phase this means that the scan failed due to a scan policy violation and the supply chain stops. For information about the CVE triage workflow, see Out of the Box Supply Chain with Testing and Scanning.

Query for vulnerabilities

Scan reports are automatically saved to the Supply Chain Security Tools - Store, and you can query them for vulnerabilities and dependencies. For example, related to open-source software (OSS) or third-party packages.

Query the tanzu-java-web-app image dependencies and vulnerabilities by running:

```
tanzu insight image get --digest DIGEST
tanzu insight image vulnerabilities --digest  DIGEST
```

Where `DIGEST` is the component version or image digest printed in the `KUBECTL GET` command.

For additional information and examples, see Tanzu Insight plug-in overview.

Congratulations! You have successfully added testing and security scanning to your application on the Tanzu Application Platform.

Take the next steps to learn about recommended supply chain security best practices and gain a powerful services journey experience on the Tanzu Application Platform by enabling several advanced use cases.

Next steps

- Configure image signing and verification in your supply chain

Add testing and scanning to your application

This topic guides you through installing the optional OOTB Supply Chain with Testing and the optional OOTB Supply Chain with Testing and Scanning.

For more information about available supply chains, see Supply chains on Tanzu Application Platform.

What you will do

- Install OOTB Supply Chain with Testing.
- Add a Tekton pipeline to the cluster and update the workload to point to the pipeline and resolve errors.
- Install OOTB Supply Chain with Testing and Scanning.

- Update the workload to point to the Tekton pipeline and resolve errors.
- Query for vulnerabilities and dependencies.

Overview

The default Out of the Box (OOTB) Supply Chain Basic and its dependencies were installed on your cluster during the Tanzu Application Platform install. As demonstrated in this guide, you can add testing and security scanning to your application. When you activate OOTB Supply Chain with Testing, it deactivates OOTB Supply Chain Basic.

The following installations also provide a sample Tekton pipeline that tests your sample application. The pipeline is configurable. Therefore, you can customize the steps to perform either additional testing or other tasks with Tekton Pipelines.

Install OOTB Supply Chain with Testing

To install OOTB Supply Chain with Testing:

1. You can activate the OOTB Supply Chain with Testing by updating your profile to use `testing` rather than `basic` as the selected supply chain for workloads in this cluster. The `tap-values.yaml` is the file used to customize the profile in `Tanzu package install tap --values-file=...`. Update `tap-values.yaml` with the following changes:

   ```
   - supply_chain: basic
   + supply_chain: testing

   - ootb_supply_chain_basic:
   + ootb_supply_chain_testing:
       registry:
         server: "SERVER-NAME"
         repository: "REPO-NAME"
   ```

   Where:

   - `SERVER-NAME` is the name of your server.
   - `REPO-NAME` is the name of the image repository that hosts the container images.

2. Update the installed profile by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION-NUMBER --
   values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.

Tekton pipeline config example

In this section, a Tekton pipeline is added to the cluster. In the next section, the workload is updated to point to the pipeline and resolve any current errors.

Note

Developers can perform this step because they know how their application must be tested. The operator can also add the Tekton pipeline to a cluster before the developer gets access.

To add the Tekton pipeline to the cluster, apply the following YAML to the cluster:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  namespace: DEVELOPER-NAMESPACE
  labels:
    apps.tanzu.vmware.com/pipeline: test     # (!) required
spec:
  params:
    - name: source-url                       # (!) required
    - name: source-revision                  # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`

              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

Where `DEVELOPER-NAMESPACE` is the name of your developer namespace.

The preceding YAML puts a Tekton pipeline in the developer namespace you specify. It defines the Tekton pipeline with a single step. The step contained in the `steps` pulls the code from the repository indicated in the developers `workload` and runs the tests within the repository. The steps of the Tekton pipeline are configurable and allow the developer to add additional items needed to test their code.

There are many steps in the supply chain. In this case, the next step is an image build. Any additional steps the developer adds to the Tekton pipeline to test their code are independent of the image being built and of any subsequent steps executed in the supply chain. This independence gives the developer freedom to focus on testing their code.

The `params` are templated by Supply Chain Choreographer. Additionally, Tekton pipelines require a Tekton `pipelineRun` to execute on the cluster. Supply Chain Choreographer handles creating the `pipelineRun` dynamically each time that step of the supply chain requires execution.

Workload update

To connect the new supply chain to the workload, the workload must be updated to point at your Tekton pipeline.

1. Update the workload by running the following with the Tanzu CLI:

   ```
   tanzu apps workload apply tanzu-java-web-app \
     --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
     --sub-path tanzu-java-web-app \
     --git-branch main \
     --type web \
     --label apps.tanzu.vmware.com/has-tests=true \
     --label app.kubernetes.io/part-of=tanzu-java-web-app \
     --yes
   ```

2. After accepting the workload creation, monitor the creation of new resources by the workload by running:

```
kubectl get workload,gitrepository,pipelinerun,images.kpack,podintent,app,servi
ces.serving
```

The result is output similar to the following example that shows the objects created by Supply Chain Choreographer:

```
NAME                                      AGE
workload.carto.run/tanzu-java-web-app   109s

NAME                                                              URL
READY   STATUS                                                                AGE
gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app   https://github.com/
vmware-tanzu/application-accelerator-samples   True    Fetched revision: main/8
72ff44c8866b7805fb2425130edb69a9853bfdf   109s

NAME                                                SUCCEEDED   REASON      START
TIME   COMPLETIONTIME
pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb   True        Succeeded   104s
77s

NAME                                  LATESTIMAGE
READY
image.kpack.io/tanzu-java-web-app   10.188.0.3:5000/foo/tanzu-java-web-app@sha2
56:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c   True

NAME                                                                  READY   REASON
AGE
podintent.conventions.carto.run/tanzu-java-web-app   True             7s

NAME                                        DESCRIPTION           SINCE-DEPLOY
AGE
app.kappctrl.k14s.io/tanzu-java-web-app   Reconcile succeeded   1s
2s

NAME                                                URL
LATESTCREATED              LATESTREADY                READY     REASON
service.serving.knative.dev/tanzu-java-web-app   http://tanzu-java-web-app.deve
loper.example.com   tanzu-java-web-app-00001   tanzu-java-web-app-00001   Unkno
wn   IngressNotConfigured
```

Install OOTB Supply Chain with Testing and Scanning

Prerequisites

- Both the Scan Controller and the default Grype scanner must be installed for scanning. Refer to the verify installation steps later in the topic.

Note

When leveraging both Tanzu Build Service and Grype in your Tanzu Application Platform supply chain, you can receive enhanced scanning coverage for the languages and frameworks with check marks in the column “Extended Scanning Coverage using Anchore Grype” on the Language and Framework Support Table.

- Add the necessary configuration to enable CVE scan results in the Tanzu Application Platform GUI. This configuration allows the Supply Chain Choreographer Tanzu Application Platform GUI plug-in to retrieve metadata about project packages and their vulnerabilities.

To install OOTB Supply Chain with Testing and Scanning:

1. Supply Chain Security Tools (SCST) - Scan is installed as part of the Tanzu Application Platform profiles. Verify that both Scan Controller and Grype Scanner are installed by running:

   ```
   tanzu package installed get scanning -n tap-install
   tanzu package installed get grype -n tap-install
   ```

   If the packages are not already installed, follow the steps in Supply Chain Security Tools - Scan to install the required scanning components.

   During installation of the Grype Scanner, sample ScanTemplates are installed into the `default` namespace. If the workload is deployed into another namespace, these sample ScanTemplates must also be present in the other namespace. One way to accomplish this is to install Grype Scanner again and provide the namespace in the values file.

   A ScanPolicy is required and must be in the required namespace. A sample ScanPolicy is provided as follows to block a supply chain when CVEs with critical, high, and unknown ratings are found using `notAllowedSeverities := ["Critical","High","UnknownSeverity"]`. You can also configure the supply chain to use your own custom policies and apply exceptions when you want to ignore certain CVEs. See Out of the Box Supply Chain with Testing and Scanning. To apply the sample ScanPolicy, you can either add the namespace flag to the kubectl command or add the namespace text box to the template by running:

   ```
   kubectl apply -f - -o yaml << EOF
   ---
   apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
   kind: ScanPolicy
   metadata:
     name: scan-policy
     labels:
       'app.kubernetes.io/part-of': 'enable-in-gui'
   spec:
     regoFile: |
       package main

       # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
   ownSeverity"
       notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
       ignoreCves := []

       contains(array, elem) = true {
         array[_] = elem
       } else = false { true }

       isSafe(match) {
         severities := { e | e := match.ratings.rating.severity } | { e | e := mat
   ch.ratings.rating[_].severity }
         some i
         fails := contains(notAllowedSeverities, severities[i])
         not fails
       }

       isSafe(match) {
         ignore := contains(ignoreCves, match.id)
         ignore
   ```

```
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
EOF
```

2. (optional) The Tanzu Application Platform profiles install the Supply Chain Security Tools - Store package by default. To persist and query the vulnerability results post-scan, confirm it is installed by running:

   ```
   tanzu package installed get metadata-store -n tap-install
   ```

   If the package is not installed, follow the installation instructions at Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles.

3. Update the profile to use the supply chain with testing and scanning by updating `tap-values.yaml` (the file used to customize the profile in `tanzu package install tap --values-file=...`) with the following changes:

   ```
   - supply_chain: testing
   + supply_chain: testing_scanning

   - ootb_supply_chain_testing:
   + ootb_supply_chain_testing_scanning:
       registry:
         server: "<SERVER-NAME>"
         repository: "<REPO-NAME>"
   ```

4. Update the `tap` package:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION-NUMBER --
   values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.

Workload update

To connect the new supply chain to the workload, update the workload to point to your Tekton pipeline:

1. Update the workload by running the following using the Tanzu CLI:

   ```
   tanzu apps workload apply tanzu-java-web-app \
     --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
     --sub-path tanzu-java-web-app \
     --git-branch main \
     --type web \
     --label apps.tanzu.vmware.com/has-tests=true \
     --label app.kubernetes.io/part-of=tanzu-java-web-app \
     --yes
   ```

2. After accepting the workload creation, view the new resources that the workload created by running:

```
kubectl get workload,gitrepository,sourcescan,pipelinerun,images.kpack,imagesca
n,podintent,app,services.serving
```

The following is an example output, which shows the objects that Supply Chain Choreographer created:

```
NAME                                      AGE
workload.carto.run/tanzu-java-web-app   109s

NAME                                                                URL
READY   STATUS                                                                  AGE
gitrepository.source.toolkit.fluxcd.io/tanzu-java-web-app   https://github.com/
vmware-tanzu/application-accelerator-samples   True    Fetched revision: main/8
72ff44c8866b7805fb2425130edb69a9853bfdf   109s

NAME                                                                PHASE       SCAN
NEDREVISION                              SCANNEDREPOSITORY
AGE    CRITICAL   HIGH   MEDIUM   LOW   UNKNOWN   CVETOTAL
sourcescan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app   Completed   1878
50b39b754e425621340787932759a0838795   https://github.com/vmware-tanzu/applicat
ion-accelerator-samples   90s

NAME                                                SUCCEEDED   REASON      START
TIME   COMPLETIONTIME
pipelinerun.tekton.dev/tanzu-java-web-app-4ftlb   True        Succeeded   104s
77s

NAME                                LATESTIMAGE
READY
image.kpack.io/tanzu-java-web-app   10.188.0.3:5000/foo/tanzu-java-web-app@sha2
56:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b896546e005f1efd98fbc4e79b7552c   True

NAME                                                                PHASE       SCANN
EDIMAGE
AGE   CRITICAL   HIGH   MEDIUM   LOW   UNKNOWN   CVETOTAL
imagescan.scanning.apps.tanzu.vmware.com/tanzu-java-web-app   Completed   10.18
8.0.3:5000/foo/tanzu-java-web-app@sha256:1d5bc4d3d1ffeb8629fbb721fcd1c4d28b8965
46e005f1efd98fbc4e79b7552c   14s

NAME                                                                  READY   REASON
AGE
podintent.conventions.carto.run/tanzu-java-web-app   True             7s

NAME                                      DESCRIPTION           SINCE-DEPLOY
AGE
app.kappctrl.k14s.io/tanzu-java-web-app   Reconcile succeeded   1s
2s

NAME                                                URL
LATESTCREATED              LATESTREADY                READY     REASON
service.serving.knative.dev/tanzu-java-web-app   http://tanzu-java-web-app.deve
loper.example.com   tanzu-java-web-app-00001   tanzu-java-web-app-00001   Unkno
wn   IngressNotConfigured
```

Important

If the source or image scan has a “Failed” phase this means that the scan failed due to a scan policy violation and the supply chain stops. For

> information about the CVE triage workflow, see Out of the Box Supply Chain with Testing and Scanning.

Query for vulnerabilities

Scan reports are automatically saved to the Supply Chain Security Tools - Store, and you can query them for vulnerabilities and dependencies. For example, related to open-source software (OSS) or third-party packages.

Query the tanzu-java-web-app image dependencies and vulnerabilities by running:

```
tanzu insight image get --digest DIGEST
tanzu insight image vulnerabilities --digest  DIGEST
```

Where `DIGEST` is the component version or image digest printed in the `KUBECTL GET` command.

For additional information and examples, see Tanzu Insight plug-in overview.

Congratulations! You have successfully added testing and security scanning to your application on the Tanzu Application Platform.

Take the next steps to learn about recommended supply chain security best practices and gain a powerful services journey experience on the Tanzu Application Platform by enabling several advanced use cases.

Next steps

- Configure image signing and verification in your supply chain

Configure image signing and verification in your supply chain

This topic guides you through configuring your Tanzu Application Platform (commonly known as TAP) supply chain to sign and verify your image builds.

What you will do

- Configure your supply chain to sign your image builds.
- Configure an admission control policy to verify image signatures before admitting pods to the cluster.

Configure your supply chain to sign and verify your image builds

1. Use Cosign to configure Tanzu Build Service to sign your container image builds. For instructions, see Configure Tanzu Build Service to sign your image builds.
2. Create a `values.yaml` file, and install the Supply Chain Security Tools - Policy Controller. For instructions, see Install Supply Chain Security Tools - Policy Controller.
3. Create a `ClusterImagePolicy` that passes Tanzu Application Platform images. It is planned for a future release for these to be signed and verifiable, but currently we recommend creating a policy to pass them:

   For example:

```
kubectl apply -f - -o yaml << EOF
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy-exceptions
spec:
  images:
  - glob: registry.example.org/myproject/*
  - glob: REPO-NAME*
  authorities:
  - static:
      action: pass
EOF
```

Where:

- `REPO-NAME` is the repository in your registry where Tanzu Build Service dependencies are stored. This is the exact same value configured in the `kp_default_repository` inside your `tap-values.yaml` or `tbs-values.yaml` files. Examples:
  - Harbor has the form `"my-harbor.io/my-project/build-service"`.
  - Docker Hub has the form `"my-dockerhub-user/build-service"` or `"index.docker.io/my-user/build-service"`.
  - Google Cloud Registry has the form `"gcr.io/my-project/build-service"`.
- Add any unsigned image that must run in your namespace to the previous policy. For example, if you add a Tekton pipeline that runs a Gradle image for testing, you need to add `glob: index.docker.io/library/gradle*` to `spec.images.glob` in the preceding code.
- Replace `registry.example.org/myproject/*` with your target registry for your Tanzu Application Platform images. If you did not relocate the Tanzu Application Platform images to your own registry during installation, use `registry.tanzu.vmware.com/tanzu-application-platform/tap-packages*`.

4. Configure and apply a `ClusterImagePolicy` resource to the cluster to verify image signatures when deploying resources. For instructions, see Create a ClusterImagePolicy resource.

   For example:

   ```
   kubectl apply -f - -o yaml << EOF
   ---
   apiVersion: policy.sigstore.dev/v1beta1
   kind: ClusterImagePolicy
   metadata:
     name: example-policy
   spec:
     images:
     - glob: registry.example.org/myproject/*
     authorities:
     - key:
         data: |
           -----BEGIN PUBLIC KEY-----
           <content ...>
           -----END PUBLIC KEY-----
   EOF
   ```

5. Enable the policy controller verification in your namespace by adding the label `policy.sigstore.dev/include: "true"` to the namespace resource.

For example:

```
kubectl label namespace YOUR-NAMESPACE policy.sigstore.dev/include=true
```

Where `YOUR-NAMESPACE` is the name of your secure namespace.

> **Note**
>
> Supply Chain Security Tools - Policy Controller only validates resources in namespaces that have chosen to opt in.

When you apply the `ClusterImagePolicy` resource, your cluster requires valid signatures for all images that match the `spec.images.glob[]` you define in the configuration. For more information about configuring an image policy, see Configuring Supply Chain Security Tools - Policy.

Next steps

- Consume services on Tanzu Application Platform

Or learn more about Supply Chain Security Tools:

- Overview for Supply Chain Security Tools - Policy
- Configuring Supply Chain Security Tools - Policy
- Supply Chain Security Tools - Policy known issues

Generate an application with Application Accelerator

This topic guides you through how to generate a new project using Application Accelerator and how to deploy the project onto a Tanzu Application Platform (commonly known as TAP) cluster. For background information, see Application Accelerator.

Prerequisites

Before you start, complete all Getting Started prerequisites.

Generate a project using an Application Accelerator

There are multiple interfaces that you can use to generate a new project. The options are:

- Application Accelerator extension for VS Code
- Application Accelerator plug-in for IntelliJ
- Tanzu Application Platform GUI

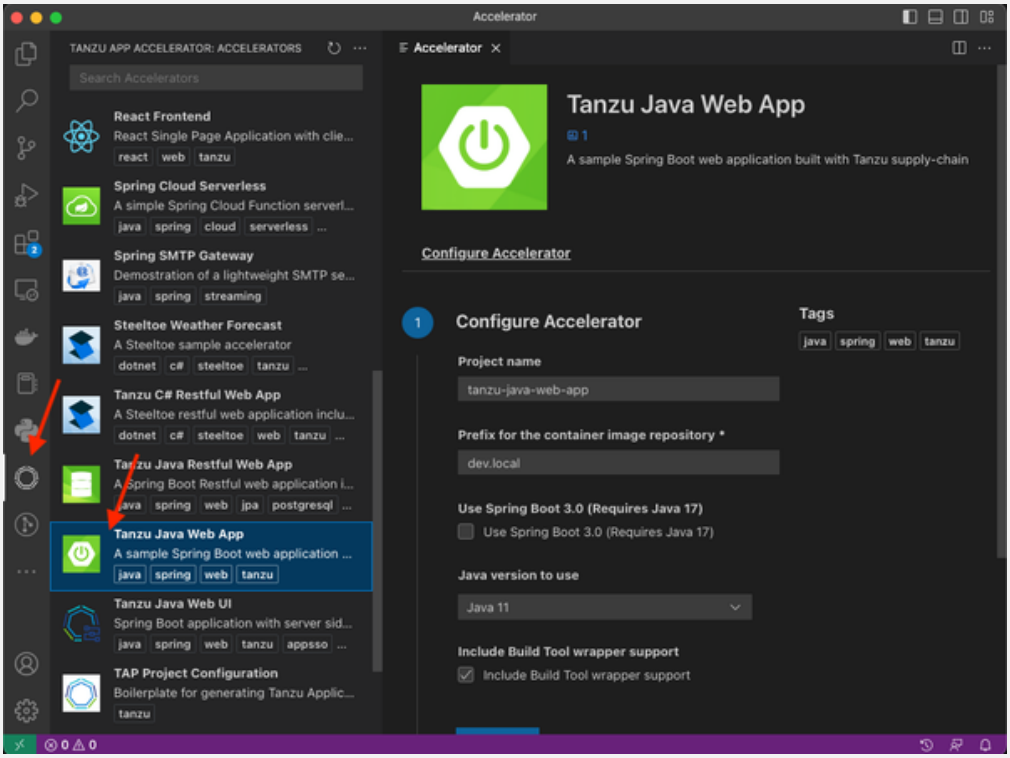
Choose one of the following tabs for how to generate and deploy applications using your selected interface. If you have already generated a project and want to skip this step, you can go to Deploying your application with Tanzu Application Platform.

VS Code
What you will do:

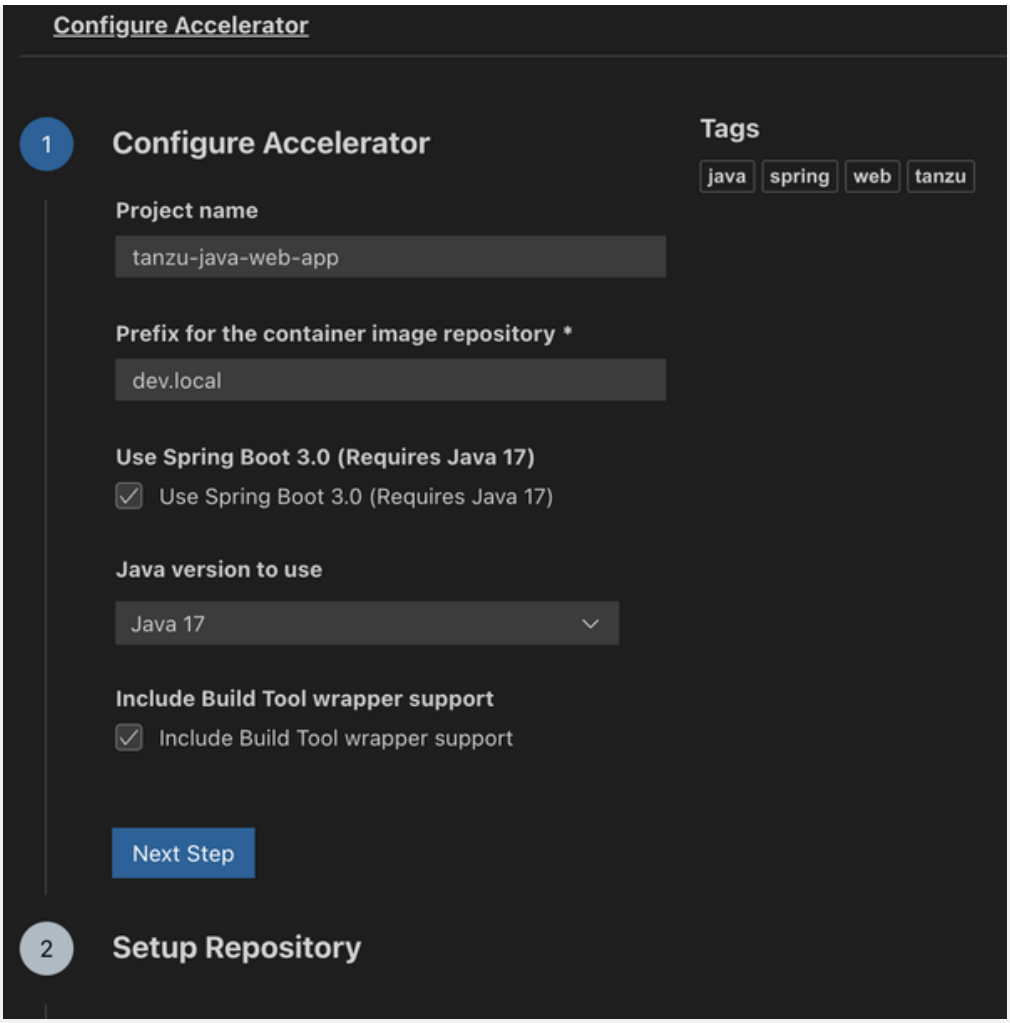
- Install the Application Accelerator extension for VS Code.
- (Optional) Provision a new GitHub repository and upload the project to the repository.
- Generate a project using an Application Accelerator.

To generate a new project using an Application Accelerator:

1. Install and configure the Application Accelerator extension for VS Code, see Application Accelerator Visual Studio Code extension.
2. Select an accelerator from the catalog. This example uses `Tanzu Java Web App`.



3. In **Configure Accelerator**, configure the accelerator as defined by your project's requirements. This example configures the project to use Spring Boot v3.0 and Java v17.

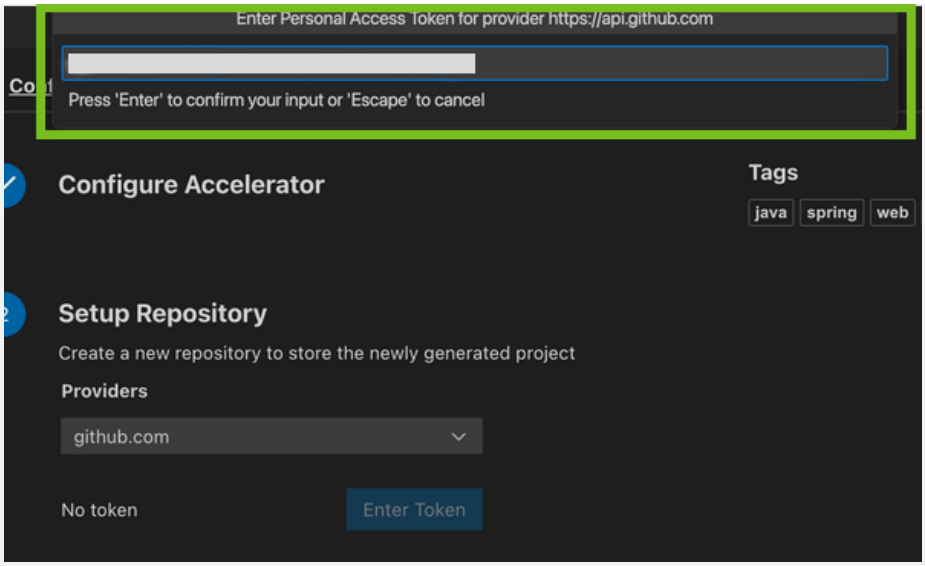


4. Click **Next Step**.
5. If your organization's Tanzu Application Platform is configured for Git repository creation, configure the **Setup Repository** step using the following sub-steps. If not, click **Skip** and go to step 5.

> **Note**
>
> For information about configuring optional Git repository creation and supported repositories, see Create an Application Accelerator Git repository during project creation.

    1. Using the **Providers** drop-down menu, select your Git provider. For example, `github.com`.
    2. After you select the provider, a dialog box appears for you to enter an API token for your Git provider. Populate the text box with your provider's API token and press Enter.

       This API key must be able to create new repositories for an organization or user. For information about how to create an API token for Git repository creation, see Creating a personal access token in the GitHub documentation.



3. In the **Owner** text box, enter the name of either the GitHub organization or user name to create the repository under.
4. In the **Repository Name** text box, enter the name of the project repository.
5. In the **Repository Branch** text box, enter the name of the default branch for the project repository. Typically, this is set to `main`.
6. Click **Next Step**.

6. In the **Review and Generate** step, verify that all the information you provided is accurate, then click **Generate Project**.
7. A dialog box appears for you to choose a location for the project to be stored on the local file system. Choose a directory or create a new one.
8. After the project has generated, a second dialog box appears for you to open the new project in a new window. Click **Yes**.
9. When opened, the project is ready for development.

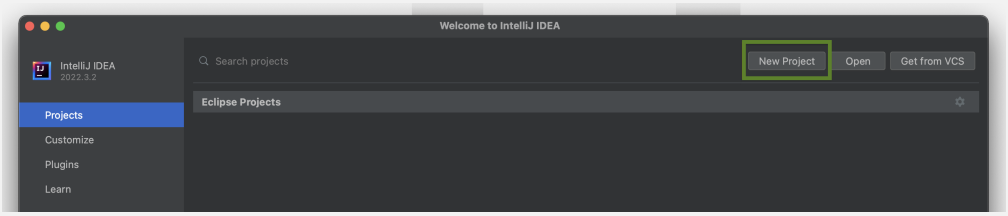
IntelliJ

What you will do:

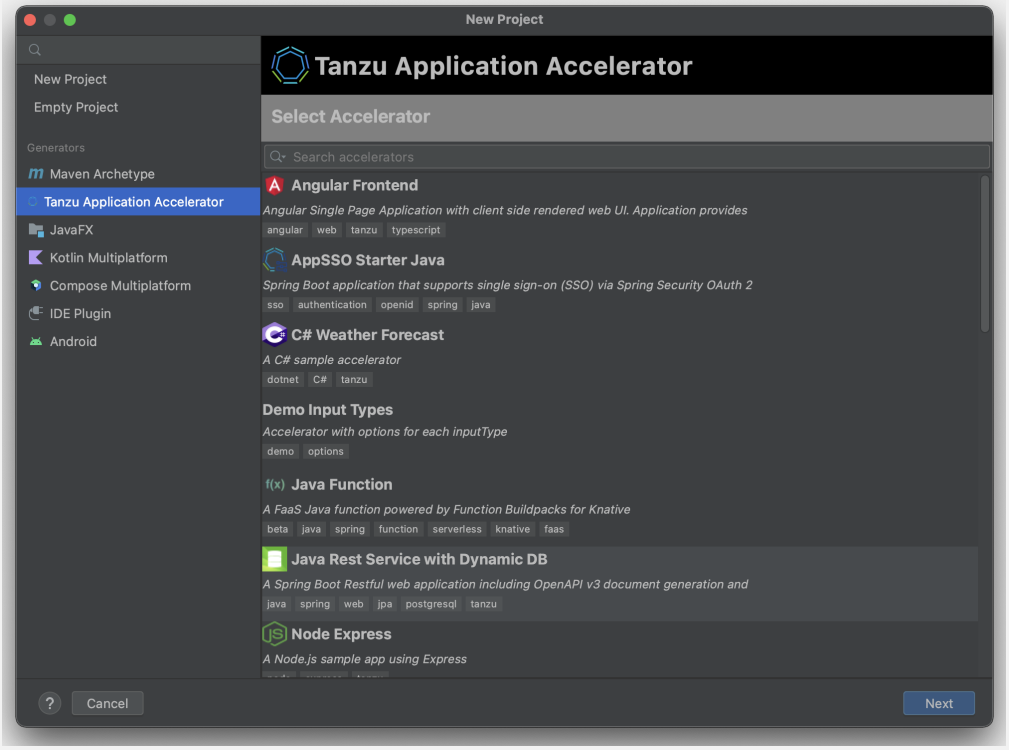
- Install the Application Accelerator plug-in for IntelliJ.
- Generate a project using an Application Accelerator.

To generate a new project using an Application Accelerator:

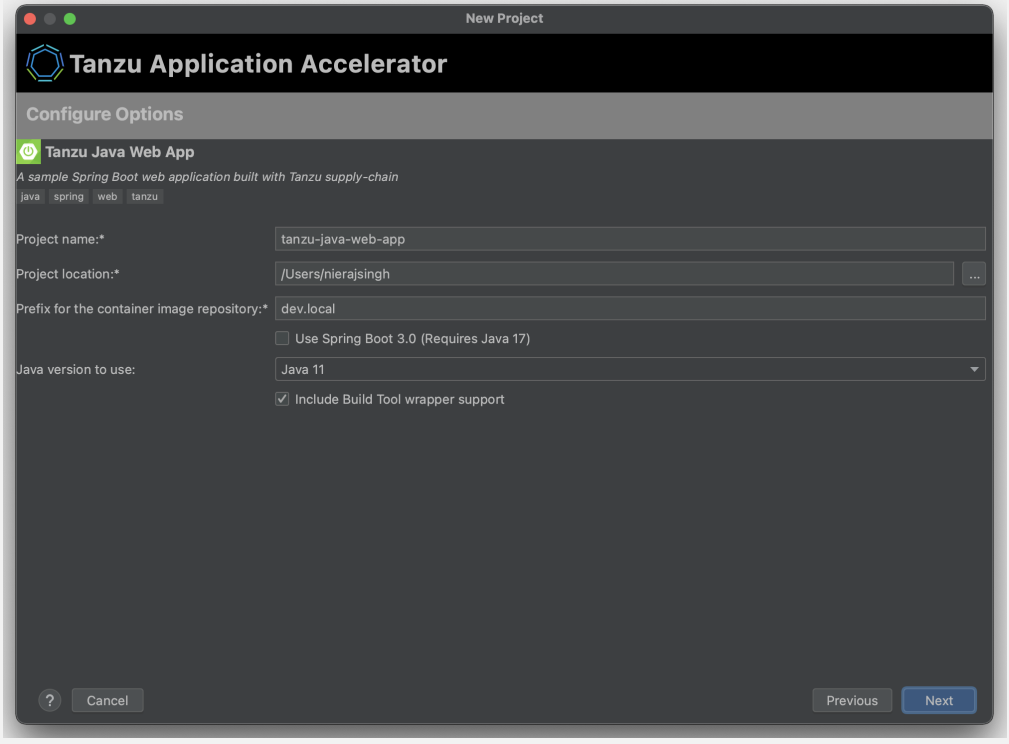
1. Install and configure the Application Accelerator plug-in for IntelliJ, see Application Accelerator plugin for IntelliJ.
2. On the Welcome to IntelliJ IDEA page, click **New Project**.



3. Click **Tanzu Application Accelerator** in the left side panel.



4. Select an accelerator from the catalog. This example uses `Tanzu Java Web App`.
5. Click **Next**.
6. In the **Configure Options** step, configure the accelerator as defined by your project's requirements.



7. Click **Next**.
8. In the **Review and Generate** step, verify that all the information provided is accurate then click **Next**.
9. After the project has generated, click **Create** to open the new project in IntelliJ.

10. When opened, the project is ready for development.

Tanzu Application Platform GUI

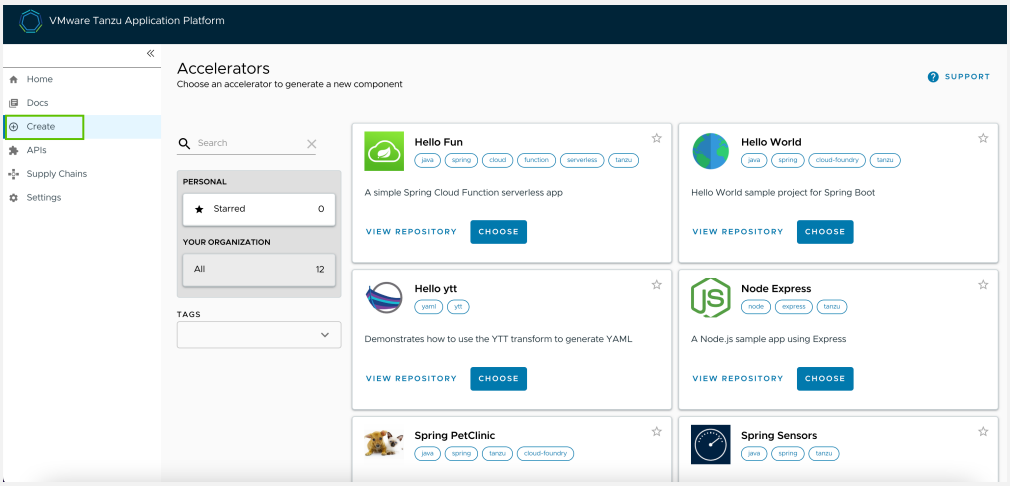
In this example, you use the `Tanzu-Java-Web-App` accelerator. You also use Tanzu Application Platform GUI. For information about connecting to Tanzu Application Platform GUI, see Access Tanzu Application Platform GUI.

What you will do:

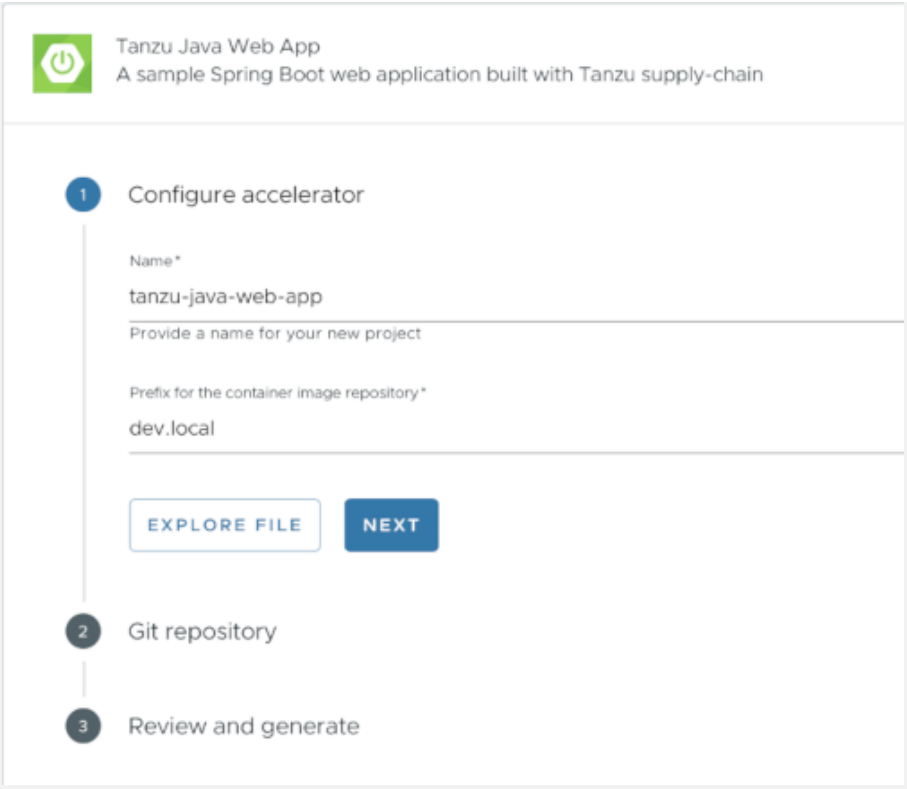
- Generate a project from an Application Accelerator.
- (Optional) Provision a new Git repository for the project.
- Upload it to your Git repository of choice.

To generate a new project using an Application Accelerator:

1. From Tanzu Application Platform GUI portal, click **Create** located on the left side of the navigation pane to see the list of available accelerators.



2. Locate the Tanzu Java Web App accelerator and click **CHOOSE**.
3. In the **Generate Accelerators** dialog box, replace the default value `dev.local` in the **prefix for container image registry** text box with the registry in the form of `SERVER-NAME/REPO-NAME`. The `SERVER-NAME/REPO-NAME` must match what was specified for `registry` as part of the installation values for `ootb_supply_chain_basic`. See the Full Profile section on Installing Tanzu Application Platform package and../install-online/install.hbs.md profiles.

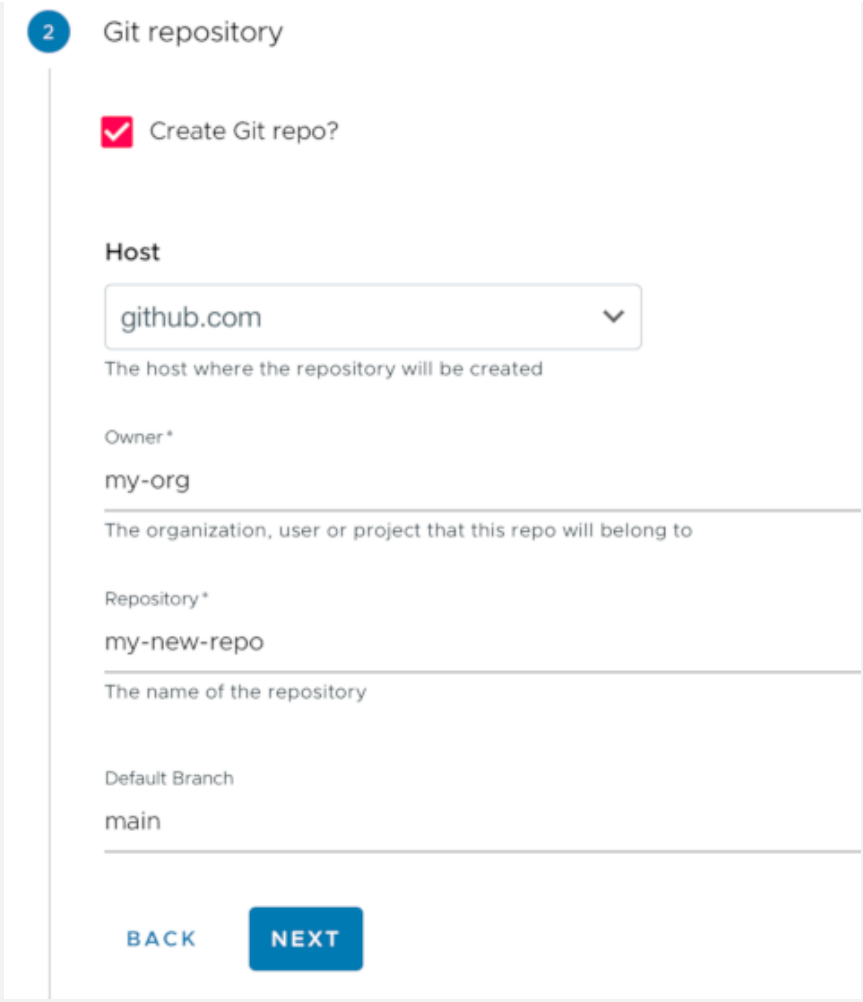


4. Click **NEXT**.
5. If your instance has optional Git repository support enabled, continue with the following sub-steps. If your instance does not support this, skip to step 5.

   Note

   For information about configuring optional Git repository creation and supported repositories, see Create an Application Accelerator Git repository during project creation.

   1. Select the **Create Git repo?** check box.
   2. Select the host Git repository provider from the **Host** drop-down menu. For example, `github.com`.
   3. Populate the **Owner** and **Repository** text boxes.



4. While you are populating the form, a dialog box appears asking for permission to provision Git repositories. Follow the prompts and continue.
5. Click **NEXT**.

6. Verify the provided information, and click **GENERATE ACCELERATOR**.
7. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
8. After downloading the ZIP file, expand it in a workspace directory. If you did not create a Git repository in the preceding steps, follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Learn more about Application Accelerator

- For information about how to configure optional Git repository creation, see Configure in *Create an Application Accelerator Git repository during project creation*.
- For information about Application Accelerator configurations, see Configure Application Accelerator.
- For information about installing the Application Accelerator extension for Visual Studio Code, see Application Accelerator Visual Studio Code extension.
- For general accelerator troubleshooting, see Troubleshooting Application Accelerator for VMware Tanzu.

Next Steps

Now that you have generated a project that is ready for Tanzu Application Platform, learn how to quickly deploy the application on a Tanzu Application Platform cluster in Deploy an app on Tanzu Application Platform.

Generate an application with Application Accelerator

This topic guides you through how to generate a new project using Application Accelerator and how to deploy the project onto a Tanzu Application Platform (commonly known as TAP) cluster. For background information, see Application Accelerator.

Prerequisites

Before you start, complete all Getting Started prerequisites.

Generate a project using an Application Accelerator

There are multiple interfaces that you can use to generate a new project. The options are:

- Application Accelerator extension for VS Code
- Application Accelerator plug-in for IntelliJ
- Tanzu Application Platform GUI

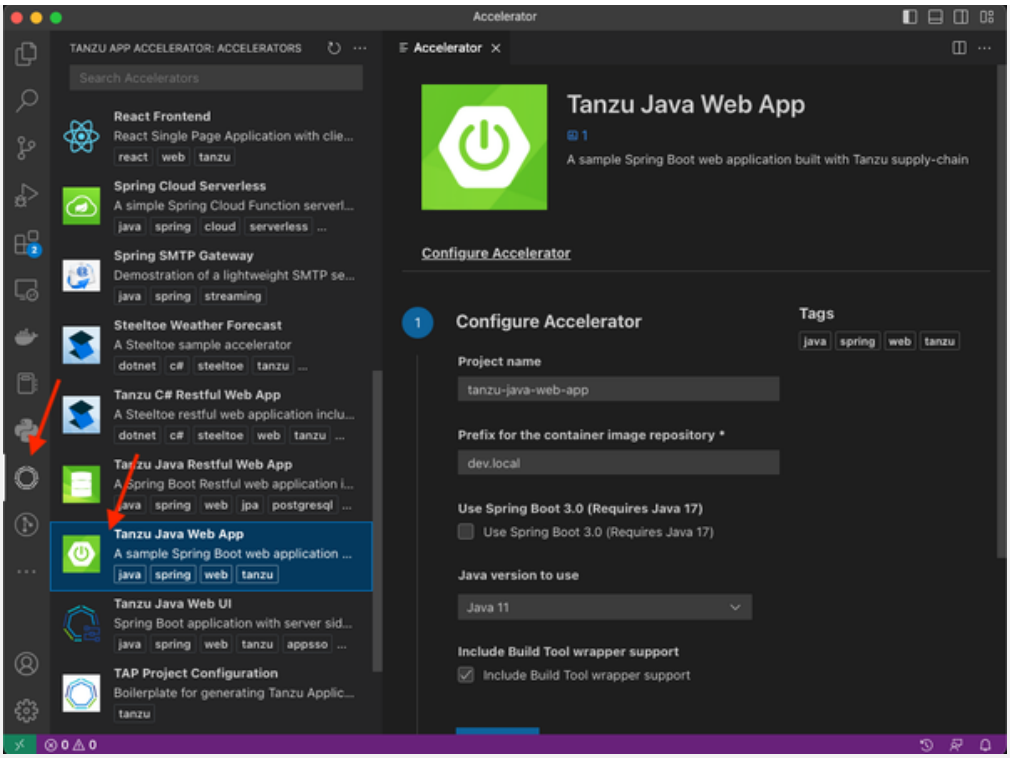
Choose one of the following tabs for how to generate and deploy applications using your selected interface. If you have already generated a project and want to skip this step, you can go to Deploying your application with Tanzu Application Platform.

VS Code
What you will do:

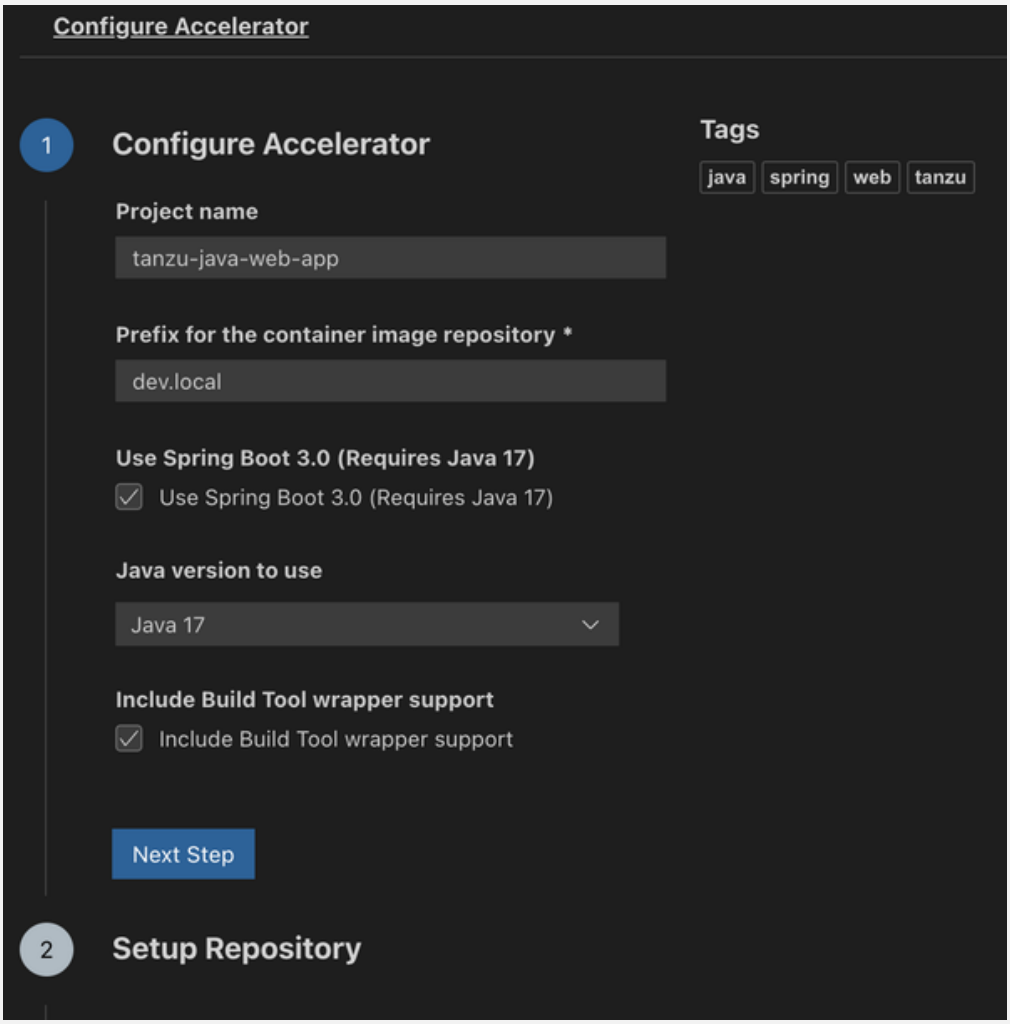
- Install the Application Accelerator extension for VS Code.
- (Optional) Provision a new GitHub repository and upload the project to the repository.
- Generate a project using an Application Accelerator.

To generate a new project using an Application Accelerator:

1. Install and configure the Application Accelerator extension for VS Code, see Application Accelerator Visual Studio Code extension.
2. Select an accelerator from the catalog. This example uses `Tanzu Java Web App`.



3. In **Configure Accelerator**, configure the accelerator as defined by your project's requirements. This example configures the project to use Spring Boot v3.0 and Java v17.

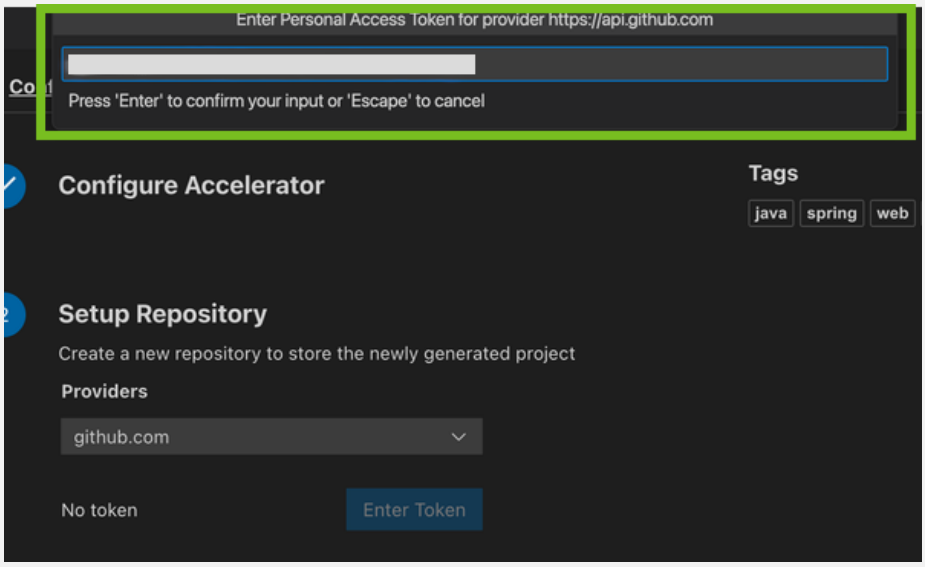


4. Click **Next Step**.
5. If your organization's Tanzu Application Platform is configured for Git repository creation, configure the **Setup Repository** step using the following sub-steps. If not, click **Skip** and go to step 5.

   > **Note**
   >
   > For information about configuring optional Git repository creation and supported repositories, see Create an Application Accelerator Git repository during project creation.

   1. Using the **Providers** drop-down menu, select your Git provider. For example, `github.com`.
   2. After you select the provider, a dialog box appears for you to enter an API token for your Git provider. Populate the text box with your provider's API token and press Enter.

      This API key must be able to create new repositories for an organization or user. For information about how to create an API token for Git repository creation, see Creating a personal access token in the GitHub documentation.

   3. In the **Owner** text box, enter the name of either the GitHub organization or user name to create the repository under.
   4. In the **Repository Name** text box, enter the name of the project repository.
   5. In the **Repository Branch** text box, enter the name of the default branch for the project repository. Typically, this is set to `main`.
   6. Click **Next Step**.
6. In the **Review and Generate** step, verify that all the information you provided is accurate, then click **Generate Project**.
7. A dialog box appears for you to choose a location for the project to be stored on the local file system. Choose a directory or create a new one.
8. After the project has generated, a second dialog box appears for you to open the new project in a new window. Click **Yes**.
9. When opened, the project is ready for development.

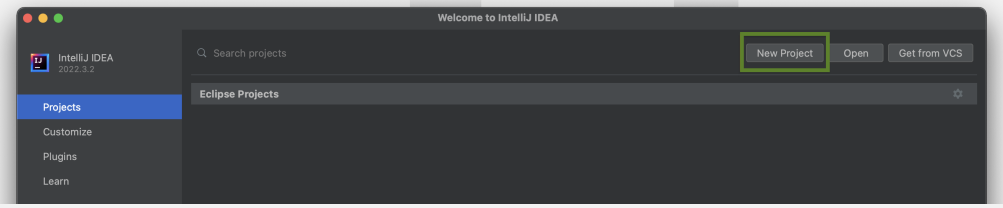
IntelliJ

What you will do:

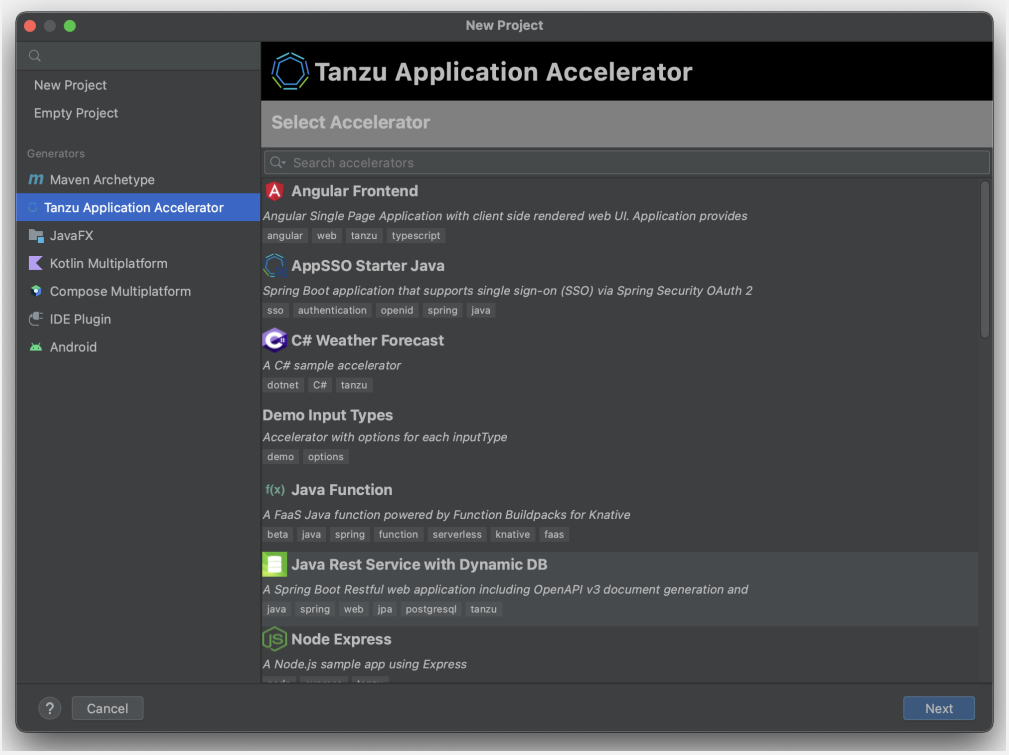
- Install the Application Accelerator plug-in for IntelliJ.
- Generate a project using an Application Accelerator.

To generate a new project using an Application Accelerator:

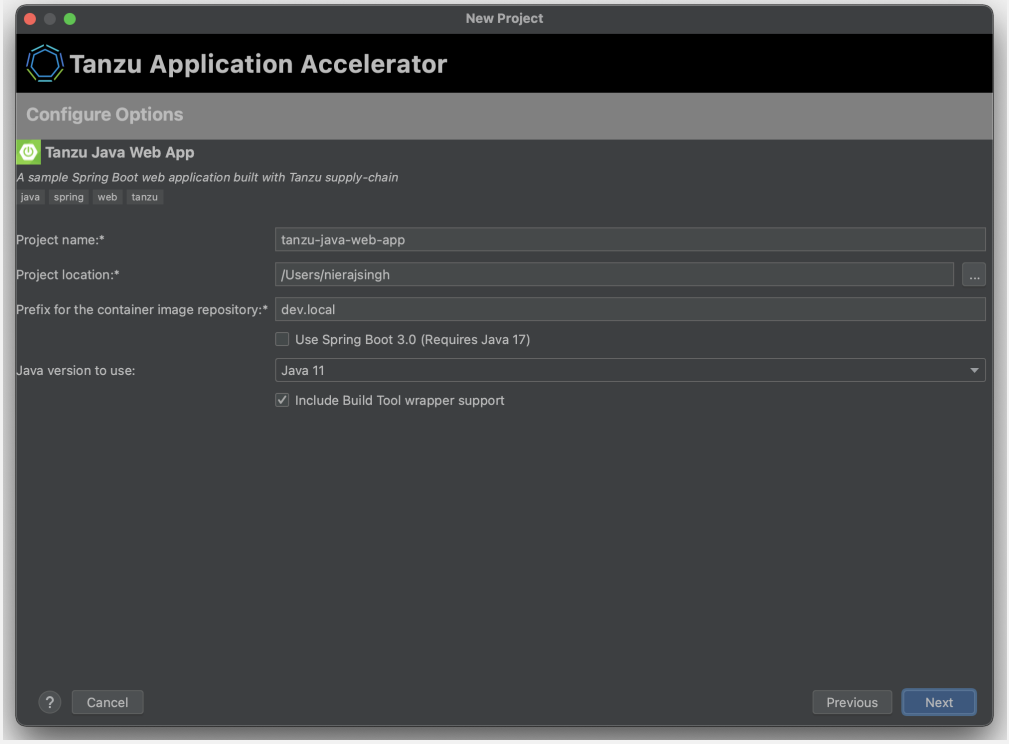
1. Install and configure the Application Accelerator plug-in for IntelliJ, see Application Accelerator plugin for IntelliJ.
2. On the Welcome to IntelliJ IDEA page, click **New Project**.



3. Click **Tanzu Application Accelerator** in the left side panel.



4. Select an accelerator from the catalog. This example uses `Tanzu Java Web App`.
5. Click **Next**.
6. In the **Configure Options** step, configure the accelerator as defined by your project's requirements.



7. Click **Next**.
8. In the **Review and Generate** step, verify that all the information provided is accurate then click **Next**.
9. After the project has generated, click **Create** to open the new project in IntelliJ.
10. When opened, the project is ready for development.

Tanzu Application Platform GUI

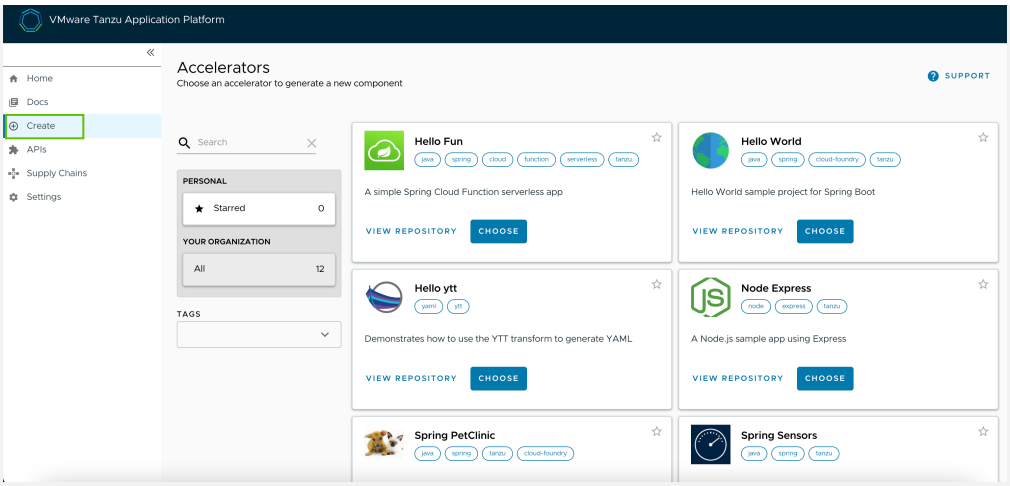
In this example, you use the `Tanzu-Java-Web-App` accelerator. You also use Tanzu Application Platform GUI. For information about connecting to Tanzu Application Platform GUI, see Access Tanzu Application Platform GUI.

What you will do:

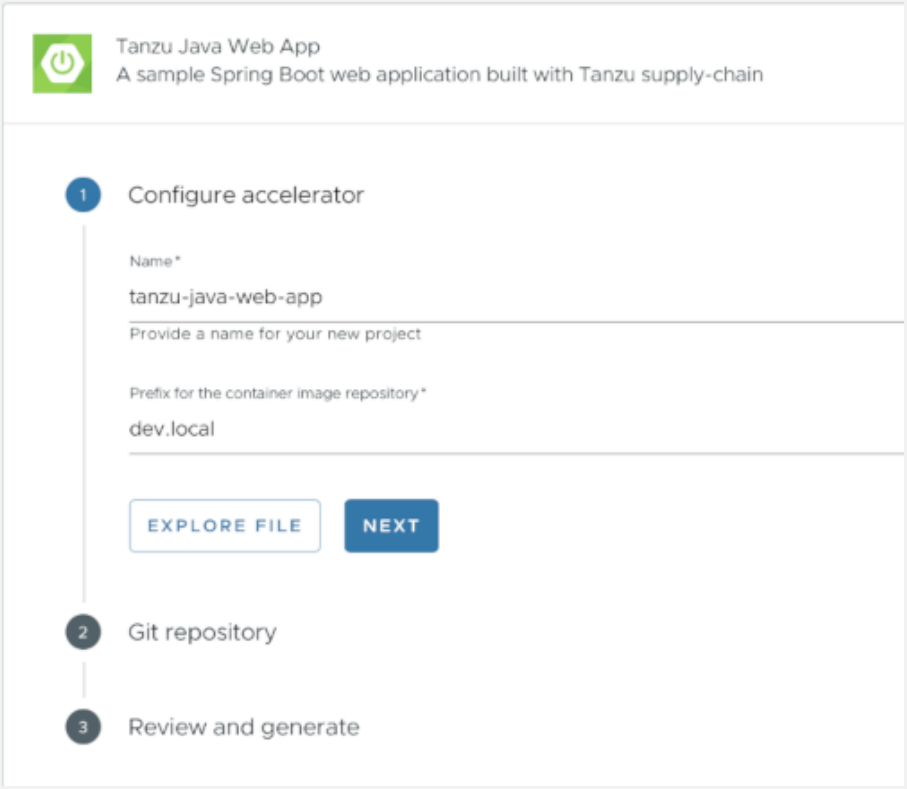
- Generate a project from an Application Accelerator.
- (Optional) Provision a new Git repository for the project.
- Upload it to your Git repository of choice.

To generate a new project using an Application Accelerator:

1. From Tanzu Application Platform GUI portal, click **Create** located on the left side of the navigation pane to see the list of available accelerators.



2. Locate the Tanzu Java Web App accelerator and click **CHOOSE**.
3. In the **Generate Accelerators** dialog box, replace the default value `dev.local` in the **prefix for container image registry** text box with the registry in the form of `SERVER-NAME/REPO-NAME`. The `SERVER-NAME/REPO-NAME` must match what was specified for `registry` as part of the installation values for `ootb_supply_chain_basic`. See the Full Profile section on Installing Tanzu Application Platform package and../install-online/install.hbs.md profiles.

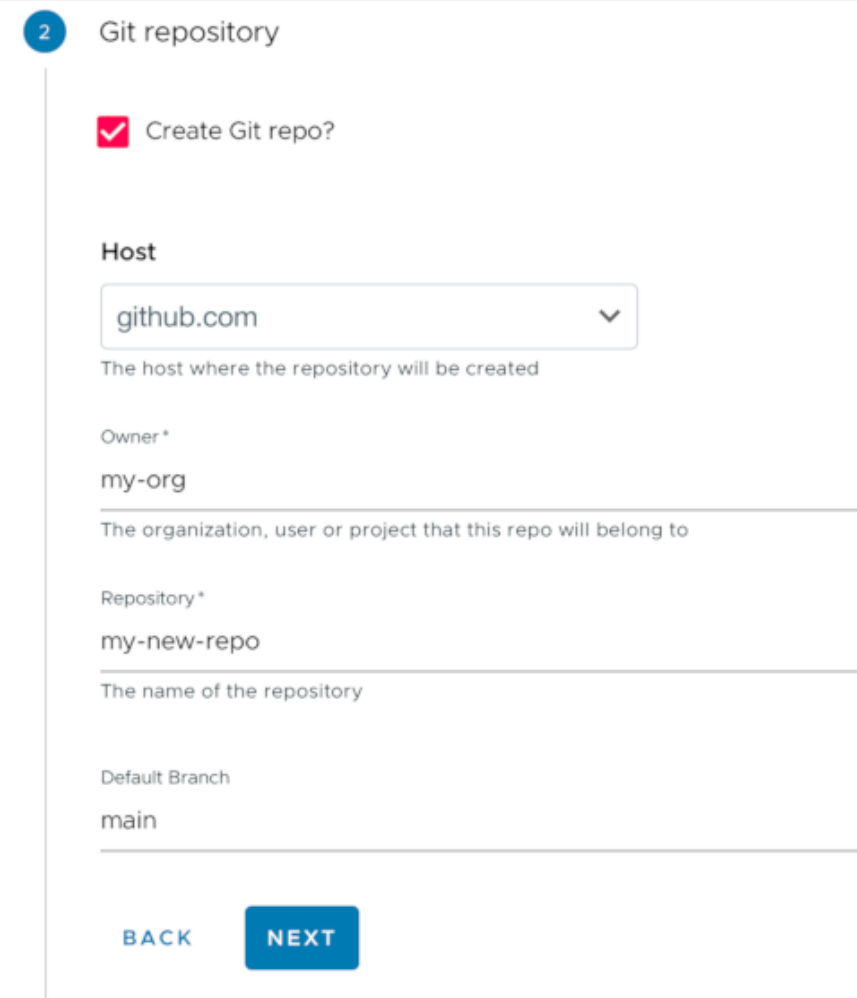


4. Click **NEXT**.
5. If your instance has optional Git repository support enabled, continue with the following sub-steps. If your instance does not support this, skip to step 5.

Note

For information about configuring optional Git repository creation and supported repositories, see Create an Application Accelerator Git repository during project creation.

1. Select the **Create Git repo?** check box.
2. Select the host Git repository provider from the **Host** drop-down menu. For example, `github.com`.
3. Populate the **Owner** and **Repository** text boxes.



4. While you are populating the form, a dialog box appears asking for permission to provision Git repositories. Follow the prompts and continue.
5. Click **NEXT**.
6. Verify the provided information, and click **GENERATE ACCELERATOR**.
7. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
8. After downloading the ZIP file, expand it in a workspace directory. If you did not create a Git repository in the preceding steps, follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Learn more about Application Accelerator

- For information about how to configure optional Git repository creation, see Configure in *Create an Application Accelerator Git repository during project creation*.
- For information about Application Accelerator configurations, see Configure Application Accelerator.
- For information about installing the Application Accelerator extension for Visual Studio Code, see Application Accelerator Visual Studio Code extension.
- For general accelerator troubleshooting, see Troubleshooting Application Accelerator for VMware Tanzu.

Next Steps

Now that you have generated a project that is ready for Tanzu Application Platform, learn how to quickly deploy the application on a Tanzu Application Platform cluster in Deploy an app on Tanzu Application Platform.

Deploy an app on Tanzu Application Platform

This topic guides you through deploying your first application on Tanzu Application Platform (commonly known as TAP) by using the Tanzu CLI, and optionally adding your application to the Tanzu Application Platform GUI software catalog.

This guide is a continuation from the previous step, Generate an application with Application Accelerator.

What you will do

- Deploy an app using the Tanzu CLI.
- View the build and runtime logs for your app.
- View the web app in your browser.
- (Optional) Add your application to Tanzu Application Platform GUI software catalog.

Prerequisites

Before you start, you must have:

- Completed all Getting Started prerequisites.
- Created a project. To do so, you can follow the steps in Generate an application with Application Accelerator.
- Created a Git repository during the project creation stage. If the project does not have an associated Git repository, create a repository and update the `workload.yaml` the repository URL and branch.

Deploy your application using the Tanzu CLI

Complete the following steps to deploy your application using the Tanzu CLI.

Prerequisites

Ensure that you meet the following prerequisites:

- Before you deploy your application using the Tanzu CLI, ensure that you have created a Git repository during the project creation stage.

- If the project does not have an associated Git repository, you must create one, and then update the `workload.yaml` with the repository URL and branch.

Procedure

1. Deploy the Tanzu Java Web App project that you generated in Generate an application with Application Accelerator by running the `tanzu apps workload create` command:

   ```
   tanzu apps workload create --file config/workload.yaml --namespace YOUR-NAMESPA
   CE
   ```

   Alternatively, you can create a workload using the command line:

   ```
   tanzu apps workload create tanzu-java-web-app \
   --git-repo GIT-REPO-URL \
   --git-branch main \
   --type web \
   --label app.kubernetes.io/part-of=tanzu-java-web-app \
   --label apps.tanzu.vmware.com/has-tests="true" \
   --yes \
   --namespace YOUR-NAMESPACE
   ```

   Where:

   - `GIT-REPO-URL` is the Git repository URL for where your project is stored. For example, `https://github.com/vmware-tanzu/my-tanzu-java-web-app-project`.
   - `YOUR-NAMESPACE` is the namespace where workloads are deployed. For example, `my-app-dev-namespace`. This depends on your organization's Tanzu Application Platform configuration. For more information, consult with your Tanzu Application Platform administrators.

   For more information, see Tanzu Apps Workload Apply.

2. View the build and runtime logs for your app by running the `get` command:

   ```
   tanzu apps workload get tanzu-java-web-app --namespace YOUR-DEVELOPER-NAMESPACE
   ```

   Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

   Note

   To watch updates in real time, prepend `watch -n1` to the `tanzu apps workload get` command to see the result update every second.

   An example of the output from an early-stage deployment looks like the following:

   ```
   Overview
       name:         tanzu-java-web-app
       type:         web
       namespace:    dev-namespace

   Source
       type:      git
       url:       https://github.com/my-organization/tanzu-java-web-app
       branch:    main

   Supply Chain
       name:    source-to-url

       NAME                READY     HEALTHY    UPDATED    RESOURCE
   ```

```
    source-provider    True      True      5s        gitrepositories.source.too
lkit.fluxcd.io/tanzu-java-web-app
    image-provider     Unknown   Unknown   5s        images.kpack.io/tanzu-java
-web-app
    config-provider    False     Unknown   8s        not found
    app-config         False     Unknown   8s        not found
    service-bindings   False     Unknown   8s        not found
    api-descriptors    False     Unknown   8s        not found
    config-writer      False     Unknown   8s        not found

Delivery
    name:   delivery-basic

    NAME              READY   HEALTHY   UPDATED   RESOURCE
    source-provider   False   False     2s        imagerepositories.source.app
s.tanzu.vmware.com/tanzu-java-web-app-delivery
    deployer          False   Unknown   5s        not found

Messages
    Workload [MissingValueAtPath]:   waiting to read value [.status.latestImag
e] from resource [images.kpack.io/tanzu-java-web-app] in namespace [dev-namespa
ce]
    Deliverable [HealthyConditionRule]:   Unable to resolve image with tag "my-
instance.azurecr.io/tap/tanzu-java-web-app-dev-namespace-bundle:0da415bc-5d79-4
d80-8ff1-0d27f42f871c" to a digest: HEAD https://my-instance.azurecr.io/v2/
    tap/tanzu-java-web-app-dev-namespace-bundle/manifests/0da415bc-5d79-4d80-8f
f1-0d27f42f871c: unexpected status code 404 Not Found (HEAD responses have no b
ody, use GET for details)

Pods
    NAME                                    READY   STATUS     RESTARTS   AGE
    tanzu-java-web-app-build-1-build-pod    0/1     Init:0/6   0          5s
```

After the workload is deployed, text similar to the following is displayed:

```
Overview
    name:        tanzu-java-web-app
    type:        web
    namespace:   dev-namespace

Source
    type:     git
    url:      https://github.com/my-organization/tanzu-java-web-app
    branch:   main

Supply Chain
    name:   source-to-url

    NAME               READY   HEALTHY   UPDATED   RESOURCE
    source-provider    True    True      5m26s     gitrepositories.source.toolk
it.fluxcd.io/tanzu-java-web-app
    image-provider     True    True      4m30s     images.kpack.io/tanzu-java-w
eb-app
    config-provider    True    True      4m24s     podintents.conventions.cart
o.run/tanzu-java-web-app
    app-config         True    True      4m24s     configmaps/tanzu-java-web-ap
p
    service-bindings   True    True      4m24s     configmaps/tanzu-java-web-ap
p-with-claims
    api-descriptors    True    True      4m24s     configmaps/tanzu-java-web-ap
p-with-api-descriptors
    config-writer      True    True      4m12s     runnables.carto.run/tanzu-ja
va-web-app-config-writer

Delivery
    name:   delivery-basic
```

```
    NAME              READY   HEALTHY   UPDATED   RESOURCE
    source-provider   True    True      3m23s     imagerepositories.source.app
s.tanzu.vmware.com/tanzu-java-web-app-delivery
    deployer          True    True      3m17s     apps.kappctrl.k14s.io/tanzu-j
ava-web-app

Messages
    No messages found.

Pods
    NAME                                          READY   STATUS      RESTARTS
AGE
    tanzu-java-web-app-build-1-build-pod          0/1     Completed   0
5m25s
    tanzu-java-web-app-config-writer-p47cg-pod    0/1     Completed   0
4m24s

Knative Services
    NAME                READY   URL
    tanzu-java-web-app  Ready   https://tanzu-java-web-app.dev-namespace.apps.
my-organization.com
```

3. After the workload is built and deployed, fetch the URL of the deployed app. The URL of the web app is in the `Knative Services` section at the bottom of the output of the `tanzu apps workload get` command:

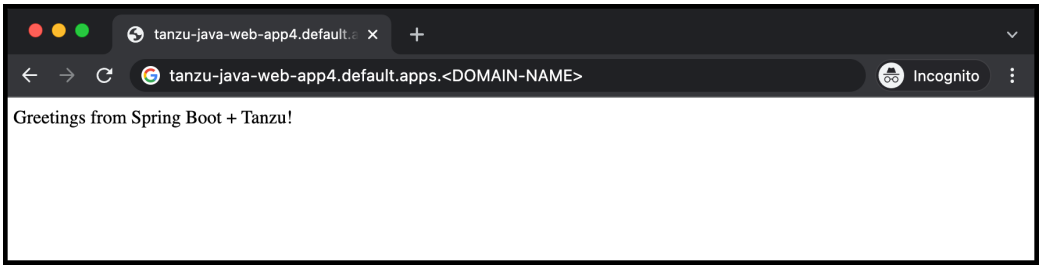
   ```
   tanzu apps workload get tanzu-java-web-app --namespace YOUR-DEVELOPER-NAMESPACE
   ```

   Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

   The output looks similar to the following:

   ```
   Knative Services
       NAME                READY   URL
       tanzu-java-web-app  Ready   https://tanzu-java-web-app.dev-namespace.apps.
   my-organization.com
   ```

4. View the web app in your browser.

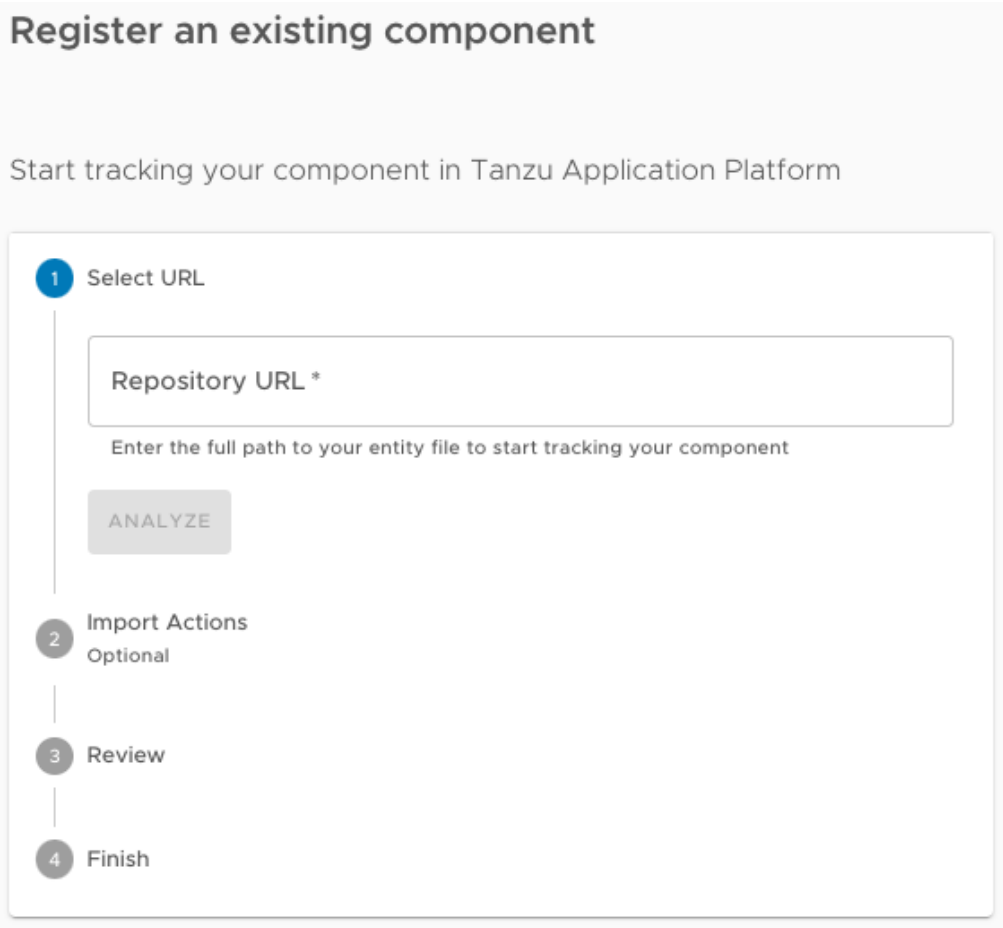


Add your application to Tanzu Application Platform GUI software catalog

1. Navigate to the home page of Tanzu Application Platform GUI and click **Home**, located on the left navigation pane.
2. Click **REGISTER ENTITY**.

   Alternatively, you can add a link for the `catalog-info.yaml` to the `tap-values.yaml` configuration file in the `tap_gui.app_config.catalog.locations` section. For more information, see Installing the Tanzu Application Platform Package and Profiles.

3. **Register an existing component** prompts you to type a repository URL. Type the link to the `catalog-info.yaml` file of the `tanzu-java-web-app` in the Git repository text box. For example, `https://github.com/USERNAME/PROJECTNAME/blob/main/catalog-info.yaml`.



4. Click **ANALYZE**.
5. Review the catalog entities to be added and click **IMPORT**.
6. Navigate back to the home page. The catalog changes and entries are visible for further inspection.

Note

If your Tanzu Application Platform GUI instance does not have a PostgreSQL database configured, you must re-register the `catalog-info.yaml` location after the instance is restarted or upgraded.

Next steps

Now that you have your application deployed on your Tanzu Application Platform cluster, the next step is to iterate on your application.

- If you are an IntelliJ user, see the Iterate on your new app using IntelliJ guide.
- If you are a Visual Studio user, see the Iterate on your new app using Visual Studio guide.
- If you are a VS Code user, see the Iterate on your new app using VS Code guide.

Iterate on your new app using Tanzu Developer Tools for IntelliJ

This topic guides you through starting to iterate on your first application on Tanzu Application Platform (commonly known as TAP). You deployed the app in the previous how-to Deploy your first application.

What you will do

- Prepare your IDE to iterate on your application.
- Live update your application to view code changes updating live on the cluster.
- Debug your application.
- Delete your application from the cluster.

Prepare your IDE to iterate on your application

In the previous Getting started how-to topic, Deploy your first application, you deployed your first application on Tanzu Application Platform. Now that you have developed a skeleton workload, you are ready to begin to iterate on your new application and test code changes on the cluster.

Tanzu Developer Tools for IntelliJ is VMware Tanzu's official IDE extension for IntelliJ. It helps you develop and receive fast feedback on your workloads running on the Tanzu Application Platform.

The IntelliJ extension enables live updates of your application while running on the cluster and allows you to debug your application directly on the cluster. For information about installing the prerequisites and the Tanzu Developer Tools for IntelliJ extension, see Install Tanzu Developer Tools for IntelliJ.

Important

Use Tilt v0.30.12 or later for the sample application.

1. Open the Tanzu Java Web App as a project within your IntelliJ IDE by selecting **File** > **Open**, then selecting the Tanzu Java Web App folder and clicking **Open**. If you don't have the Tanzu Java Web App you can obtain it by following the instructions in Generate a new project using an Application Accelerator, or from the Application Accelerator Samples GitHub page.
2. Confirm that your current Kubernetes context contains a default namespace. The Tanzu Panel, found by clicking **Tanzu Panel** at the bottom-left of the IntelliJ window, uses the default namespace associated with your current Kubernetes context to populate the workloads from the cluster.
    1. Open the Terminal by clicking **View** > **Terminal**.
    2. Ensure that your current context has a default namespace by running:

        ```
        kubectl config get-contexts
        ```

        This command returns a list of all of your Kubernetes contexts with an asterisk (`*`) in front of your current context. Verify that your current context has a namespace in the namespace column.
    3. If your current context does not have a namespace in the namespace column, run:

```
kubectl config set-context --current --namespace=YOUR-DEVELOPER-NAMESPACE
```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace value you want to assign to your current Kubernetes context.

You are now ready to iterate on your application.

Apply your application to the cluster

Apply the workload to see your application running on the cluster:

1. In the **Project** tab in IntelliJ, right-click any file under the application name `tanzu-java-web-app` and click **Tanzu** > **Apply Workload**.
2. In the dialog box enter your **Source Image**, **Local Path**, and optionally a **Namespace**.
    1. In the **Source Image** text box, provide the destination image repository to publish an image containing your workload source code.

        The source image value tells the Tanzu Developer Tools for IntelliJ extension where to publish the container image with your uncompiled source code, and what to name that image. The image must be published to a container image registry where you have write (push) access. For example, `gcr.io/myteam/tanzu-java-web-app-source`.

        Note

        See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

        For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.

    2. In the **Local Path** text box, provide the path to the directory containing the Tanzu Java Web App. The current directory is the default.

        The local path value tells the Tanzu Developer Tools for IntelliJ extension which directory on your local file system to bring into the source image. For example, dot (`.`) uses the working directory, or you can specify a full file path.

    3. (Optional) In the **Namespace** text box, provide the namespace to be associated with the workload on the cluster. If you followed the steps to Prepare your IDE to iterate on your application earlier, you do not need to enter a namespace because IntelliJ uses the namespace you associated with your context.
    4. Click the **OK** button.

The `apply workload` command runs, which opens a terminal and shows you the output of the command. The `apply workload` command can take a few minutes to deploy your application onto the cluster.

You can also use the **Tanzu Panel** to monitor your application as it's being deployed to the cluster. The **Tanzu Panel** shows information about the workloads in the namespace associated with your current Kubernetes context. On the left side, it shows the workloads in the namespace. In the center, it shows the details of the Kubernetes resources for the running workloads.

Enable Live Update for your application

Live Update allows you to save changes to your code and see those changes reflected within seconds in the workload running on the cluster.

To enable Live Update for your application:

1. Create a Run Configuration.
    1. In IntelliJ, select the **Edit Run/Debug configurations** drop-down menu at the top-right corner. Alternatively, navigate to **Run** > **Edit Configurations**.
    2. Select **Tanzu Live Update**.
    3. Select **Add new run configuration**, or click the plus icon at the top of the list.
    4. Give your new run configuration a name, for example, `Tanzu Live Update - tanzu-java-web-app`.
    5. In the **Tiltfile Path** text box, provide the path to the `Tiltfile` in the Tanzu Java Web App project directory.
    6. Select the folder icon on the right-side of the text box, go to the `Tanzu Java Web App` directory, select the `Tiltfile`, and click **Open**. The `Tiltfile` facilitates Live Update using Tilt.
    7. In the **Local Path** text box, provide the path to the directory containing the Tanzu Java Web App.

        The local path value tells the Tanzu Developer Tools for IntelliJ extension which directory on your local file system to bring into the source image.

        For example, `/Users/developer/Documents/tanzu-java-web-app`.
    8. In the **Source Image** text box, provide the destination image repository to publish an image containing your workload source code.

        The source image value tells the Tanzu Developer Tools for IntelliJ extension where to publish the container image with your uncompiled source code, and what to name that image. The image must be published to a container image registry where you have write (push) access. For example, `gcr.io/myteam/tanzu-java-web-app-source`.

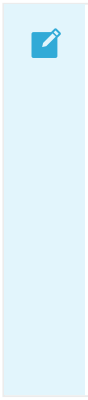
        Note

        See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

        For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.
    9. Click **Apply**, and then click the **OK** button.
2. Begin Live Updating the application on the cluster by doing one of the following:
    - In the Project tab of IntelliJ, right-click the `Tiltfile` file under the application name `tanzu-java-web-app` and click **Run 'Tanzu Live Update - tanzu-java-web-app'**.
    - Alternatively, click the **Edit Run/Debug configurations** drop-down menu in the top-right corner, select **Tanzu Live Update - tanzu-java-web-app**, and then click the green play button to the right of the **Edit Run/Debug configurations** drop-down menu.

    The **Run** tab opens and displays the output from Tanzu Application Platform and from Tilt indicating that the container is being built and deployed.

On the **Tanzu Panel** tab, the status of Live Update is reflected under the `tanzu-java-web-app` workload entry. Live update can take up to three minutes while the workload deploys and the Knative service becomes available.



Note

Depending on the type of cluster you use, you might see an error similar to the following:

```
ERROR: Stop! cluster-name might be production. If you're sure you want to deploy there, add: allow_k8s_contexts('cluster-name') to your Tiltfile. Otherwise, switch k8scontexts and restart Tilt.
```

Follow the instructions and add the line, `allow_k8s_contexts('cluster-name')` to your `Tiltfile`.

3. When the Live Update task in the **Run** tab is successful, it resolves to `Live Update Started`. Use the hyperlink at the top of the Run output following the words **Tilt started on** to view your application in your browser.
4. In the IDE, make a change to the source code. For example, in `HelloController.java`, edit the string returned to say `Hello!` and save.
5. (Optional) Build your project by clicking **Build** > **Build Project** if you do not have **Build project automatically** activated under **Preferences** > **Build, Execution, Deployment** > **Compiler**.
6. The container is updated when the logs stop streaming. Navigate to your browser and refresh the page.
7. View the changes to your workload running on the cluster.
8. Either continue making changes, or stop the Live Update process when finished. To stop Live Update navigate to the **Run** tab at the bottom left of the IntelliJ window and click the red stop icon on the left side of the screen.

Debug your application

Debug the cluster either on the application or in your local environment.

To debug the cluster:

1. Set a breakpoint in your code. For example, in `HelloController.java`, set a breakpoint on the line returning text.
2. Create a Run Configuration.
    1. In IntelliJ, select the **Edit Run/Debug configurations** drop-down menu at the top-right corner. Alternatively, navigate to **Run** > **Edit Configurations**.
    2. Select **Tanzu Debug Workload**.
    3. Select **Add new run configuration**, or click the plus icon at the top of the list.
    4. Give your new run configuration a name, for example, `Tanzu Debug Workload - tanzu-java-web-app`.
    5. In the **Workload File Path** text box, provide the path to the `workload.yaml` file in the Tanzu Java Web App project directory located at **Config** > **workload.yaml**.
    6. Select the folder icon on the right-side of the text box, navigate to the Tanzu Java Web App directory, select the `workload.yaml` file and click the **Open** button. The

`workload.yaml` provides configuration instructions about your application to the Tanzu Application Platform.

7. In the **Local Path** text box, provide the path to the directory containing the Tanzu Java Web App.

   The local path value tells the Tanzu Developer Tools for IntelliJ extension which directory on your local file system to bring into the source image. For example, `/Users/developer/Documents/tanzu-java-web-app`.

8. In the **Source Image** text box, provide the destination image repository to publish an image containing your workload source code.

   The source image value tells the Tanzu Developer Tools for IntelliJ extension where to publish the container image with your uncompiled source code, and what to name that image. The image must be published to a container image registry where you have write (push) access. For example, `gcr.io/myteam/tanzu-java-web-app-source`.

   > **Note**
   >
   > See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.
   >
   > For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.

9. (Optional) In the **Namespace** text box, provide the namespace to be associated with the workload on the cluster. If you followed the steps to Prepare your IDE to iterate on your application, you do not need to enter a namespace because IntelliJ uses the namespace you associated with your context.

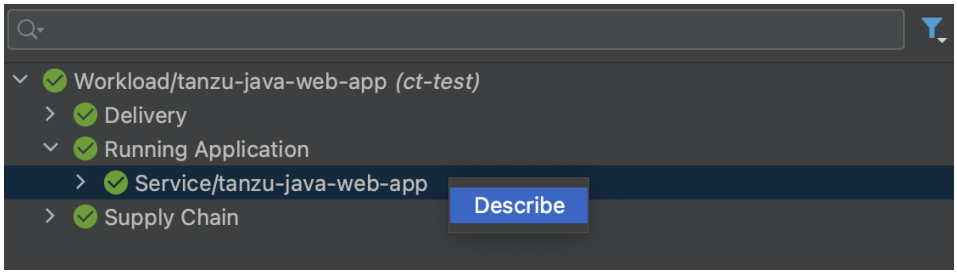
10. Click **Apply**, and then click **OK**.

3. Apply your application to the cluster.

4. Obtain the URL for your workload:

   1. In the center panel of the **Tanzu Panel** go to **Workload/tanzu-java-web-app** > **Running Application** > **Service/tanzu-java-web-app**.

   2. Right-click the `Service/tanzu-java-web-app` entry and select **Describe**.

   3. In the resulting output, copy the value after **Status** > **URL:** that begins with `https://tanzu-java-web-app...`. Make sure you copy the value from **Status** > **URL:** and *not* the value under **Status** > **Address** > **URL**.

```
Status:
  Address:
    URL:  http://tanzu-java-web-app.ct-test.svc.cluster.local
  Conditions:
    Last Transition Time:        2023-03-30T23:56:50Z
    Status:                      True
    Type:                        ConfigurationsReady
    Last Transition Time:        2023-03-30T23:56:52Z
    Status:                      True
    Type:                        Ready
    Last Transition Time:        2023-03-30T23:56:52Z
    Status:                      True
    Type:                        RoutesReady
  Latest Created Revision Name:  tanzu-java-web-app-00001
  Latest Ready Revision Name:    tanzu-java-web-app-00001
  Observed Generation:           1
  Traffic:
    Latest Revision:  true
    Percent:          100
    Revision Name:    tanzu-java-web-app-00001
  URL:                https://tanzu-java-web-app.ct-test.desktop-dev-15-build-38.tlc.dev
Events:               <none>
```

4. Open your web browser and paste the URL you copied to access your workload.

5. In the Project tab of IntelliJ, right-click the `workload.yaml` file under the application name `tanzu-java-web-app` and select **Run 'Tanzu Debug Workload - tanzu-java-web-app'** to begin debugging the application on the cluster.

    1. Alternatively, select the **Edit Run/Debug configurations** drop-down menu in the top-right corner, select **Tanzu Debug Workload - tanzu-java-web-app**, and then click the green debug button to the right of the **Edit Run/Debug configurations** drop-down menu.

6. The Debug tab opens and displays a message that it has connected.

7. In your web browser, reload your workload. IntelliJ opens to show your breakpoint.

8. You can now use the resume program action, or stop debugging, in the **Debug** tab.

Delete your application from the cluster

You can use the delete action to remove your application from the cluster as follows:

1. In the **Project** tab, right-click any file under the application name `tanzu-java-web-app` and select **Tanzu** > **Delete Workload**.

2. Alternatively, right-click `tanzu-java-web-app` in the **TANZU WORKLOADS** panel and select **Delete Workload**.

3. In the confirmation dialog box that appears, click **OK** to delete the application from the cluster.

Next steps

- Consume services on Tanzu Application Platform

Iterate on your new app using Tanzu Developer Tools for Visual Studio

This topic guides you through starting to iterate on your first application on Tanzu Application Platform (commonly known as TAP). You deployed the app in the previous how-to Deploy your first

application.

What you will do

- Prepare to iterate on your application.
  - Prepare your project to support Live Update.
  - Prepare your IDE to iterate on your application.
- Apply your application to the cluster.
- Live update your application to view code changes updating live on the cluster.
- Debug your application.
- Monitor your running application on the Application Live View UI.
- Delete your application from the cluster.

Prepare to iterate on your application

In the previous Getting started how-to topic, Deploy your first application, you deployed your first application on Tanzu Application Platform. Now that you have developed a skeleton workload, you are ready to begin to iterate on your new application and test code changes on the cluster.

Tanzu Developer Tools for Visual Studio is VMware Tanzu's official IDE extension for Visual Studio. It helps you develop and receive fast feedback on your workloads running on the Tanzu Application Platform.

The Visual Studio extension enables live updates of your application while running on the cluster and allows you to debug your application directly on the cluster.

For information about installing the prerequisites and the Tanzu Developer Tools for Visual Studio extension, see Install Tanzu Developer Tools for Visual Studio.

Important

Use Tilt v0.30.12 or later for the sample application.

To prepare to iterate on your application, you must:

1. Prepare your project to support Live Update
2. Set up the IDE

Prepare your project to support Live Update

Tanzu Live Update uses Tilt. This requires a suitable `Tiltfile` to exist at the root of your project.

Your `Tiltfile` must be similar to the following:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='your-registry.io/project/csharp-weat
herforecast-source')
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')
NAME = os.getenv("NAME", default='sample-app')

k8s_custom_deploy(
    NAME,
    apply_cmd="tanzu apps workload apply -f config/workload.yaml --update-strategy rep
lace --debug --live-update" +
              " --local-path " + LOCAL_PATH +
```

```
                " --namespace " + NAMESPACE +
                " --yes --output yaml",
    delete_cmd="tanzu apps workload delete " + NAME + " --namespace " + NAMESPACE + "
--yes",
    deps=['./bin'],
    container_selector='workload',
    live_update=[
      sync('./bin/Debug/net6.0', '/workspace')
    ]
)

k8s_resource('tanzu-java-web-app', port_forwards=["8080:8080"],
             extra_pod_selectors=[{'carto.run/workload-name': 'sample-app', 'app.kubern
etes.io/component': 'run'}])
```

Set up the IDE

After verifying your project has the required `Tiltfile`, you are ready to set up your development environment.

1. Open the Weather Forecast solution in Visual Studio by selecting **File** > **Open** > **Project/Solution...**. If you don't have the Weather Forecast app you can obtain it by following the instructions in Generate an application with Application Accelerator, or from the Application Accelerator Samples GitHub page.

You are now ready to iterate on your application.

Apply your application to the cluster

Apply the workload to see your application running on the cluster:

1. In **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Apply Workload**.
2. In the dialog box, enter the following:
    1. In the **Local Path** text box, provide the path to the directory containing the Weather Forecast app. The current directory is the default.

       The local path value tells the Tanzu Developer Tools for Visual Studio extension which directory on your local file system to bring into the source image. For example, dot (`.`) uses the working directory, or you can specify a full file path.
    2. In the **Namespace** text box, provide the namespace to be associated with the workload on the cluster.
    3. (Optional) In the **Source Image** text box, provide the destination image repository to publish the image containing your workload source code.

       The source image value tells the Tanzu Developer Tools for Visual Studio extension where to publish the container image with your uncompiled source code, and what to name that image. The image must be published to a container image registry where you have write (push) access. For example, `gcr.io/myteam/weather-forecast-source`.

       Note

       See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

> For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.

4. Click the **OK** button.

The `apply workload` command runs and opens a an output window in which you can monitor the output of the command. The `apply workload` command can take a few minutes to deploy your application onto the cluster.

Enable Live Update for your application

Live Update allows you to save changes to your code and see those changes reflected within seconds in the workload running on the cluster.

To enable Live Update for your application:

1. In **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Start Live Update**.
2. Live update can take up to three minutes while the workload deploys and the Knative service becomes available.

> **Note**
>
> Depending on the type of cluster you use, you might see an error similar to the following:
>
> ```
> ERROR: Stop! cluster-name might be production. If you're sure you want to deploy there, add: allow_k8s_contexts('cluster-name') to your Tiltfile. Otherwise, switch k8scontexts and restart Tilt.
> ```
>
> Follow the instructions and add the line, `allow_k8s_contexts('cluster-name')` to your `Tiltfile`.

3. In the IDE, make a change to the source code.
4. Build your project.
5. The container is updated when the logs stop streaming. Go to your browser and refresh the page.
6. View the changes to your workload running on the cluster.
7. Either continue making changes, or stop the Live Update process when finished. To stop Live Update, in **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Stop Live Update**.

Debug your application

Debug the cluster either on the application or in your local environment.

To start debugging the cluster:

1. Set a breakpoint in your code.
2. Apply your application to the cluster.
3. In **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Debug Workload**.

To stop debugging the cluster:

1. In main, click **Debug** > **Detach All**

Delete your application from the cluster

You can use the delete action to remove your application from the cluster as follows:

1. In **Solution Explorer**, right-click any file under the application name and click **Tanzu** > **Delete Workload**.
2. In the confirmation dialog box that appears, click **OK** to delete the application from the cluster.

Next steps

- Consume services on Tanzu Application Platform

Iterate on your new app using Tanzu Developer Tools for VS Code

This topic guides you through starting to iterate on your first application on Tanzu Application Platform (commonly known as TAP). You deployed the app in the previous how-to Deploy your first application.

What you will do

- Prepare your IDE to iterate on your application.
- Apply your application to the cluster.
- Live update your application to view code changes updating live on the cluster.
- Debug your application.
- Monitor your running application on the Application Live View UI.
- Delete your application from the cluster.

Prepare your IDE to iterate on your application

In the previous Getting started how-to topic, Deploy your first application, you deployed your first application on Tanzu Application Platform. Now that you have developed a skeleton workload, you are ready to begin to iterate on your new application and test code changes on the cluster.

Tanzu Developer Tools for VS Code is VMware Tanzu's official IDE extension for VS Code. It helps you develop and receive fast feedback on your workloads running on the Tanzu Application Platform.

The VS Code extension enables live updates of your application while running on the cluster and allows you to debug your application directly on the cluster. For information about installing the prerequisites and the Tanzu Developer Tools for VS Code extension, see Install Tanzu Developer Tools for your VS Code.

Important

Use Tilt v0.30.12 or a later version for the sample application.

1. Open the Tanzu Java Web App as a project within your VS Code IDE by clicking **File** > **Open Folder**, select the Tanzu Java Web App folder and click **Open**.

   If you don't have the Tanzu Java Web App you can obtain it by following the instructions in Generate a new project using an Application Accelerator, or from the Application Accelerator Samples GitHub page.

2. To ensure that your extension assists you with iterating on the correct project, configure its settings as follows:

   1. In Visual Studio Code, navigate to **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools**.
   2. In the **Local Path** text box, provide the path to the directory containing the Tanzu Java Web App. The current directory is the default.

      The local path value tells the Tanzu Developer Tools for VS Code extension which directory on your local file system to bring into the source image. For example, dot (`.`) uses the working directory, or you can specify a full file path.

   3. In the **Source Image** text box, provide the destination image repository to publish an image containing the workload source code.

      The source image value tells the Tanzu Developer Tools for VS Code extension where to publish the container image with your uncompiled source code, and what to name that image. The image must be published to a container image registry where you have write (push) access. For example, `gcr.io/myteam/tanzu-java-web-app-source`.

      Note

      See the documentation for the registry you're using to find out which steps are necessary to authenticate and gain push access.

      For example, if you use Docker, see the Docker documentation, or if you use Harbor, see the Harbor documentation.

      For troubleshooting failed registry authentication, see Troubleshoot using Tanzu Application Platform

3. Confirm that your current Kubernetes context has a namespace associated with it. The **TANZU WORKLOADS** section of the **Explorer** view in the left Side Bar uses the namespace associated with your current Kubernetes context to populate the workloads from the cluster.

   1. Open the Terminal by clicking **View** > **Terminal**.
   2. Ensure your current context has a default namespace by running:

      ```
      kubectl config get-contexts
      ```

      This command returns a list of all of your Kubernetes contexts with an asterisk (`*`) in front of your current context. Verify that your current context has a namespace in the namespace column.

   3. If your current context does not have a namespace in the namespace column, run:

      ```
      kubectl config set-context --current --namespace=YOUR-DEVELOPER-NAMESPACE
      ```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace value you want to assign to your current Kubernetes context.

You are now ready to iterate on your application.

Apply your application to the cluster

Apply the workload to see your application running on the cluster by doing one of the following:

- In the **Explorer** view in the left Side Bar, right-click any file under the application name `tanzu-java-web-app` and click **Tanzu: Apply Workload** to begin applying the workload to the cluster.
- Alternatively, use the Command Palette, ⇧⌘P on Mac and Ctrl+Shift+P on Windows or **View** > **Command Palette**, to run the `Tanzu: Apply Workload` command.

The `apply workload` command runs, which opens a terminal and shows you the output of the workload apply.

You can also monitor your application as it's being deployed to the cluster using the **TANZU ACTIVITY** tab in the Panel at the bottom of VS Code. The **TANZU ACTIVITY** tab shows the details of the Kubernetes resources for the workloads running in the namespace associated with your current Kubernetes context.

To view the **TANZU ACTIVITY** tab, open the Panel at the bottom of VS Code (**View** > **Appearance** > **Panel**) and then click the **TANZU ACTIVITY** tab. The apply workload command can take a few minutes to deploy your application onto the cluster. After complete, you can see the workload running in the **TANZU WORKLOADS** section of the **Explorer** view in the left Side Bar.

Enable Live Update for your application

Live Update allows you to save changes to your code and see those changes reflected within seconds in the workload running on the cluster.

To enable Live Update for your application:

1. To begin Live Updating the workload on the cluster, do one of the following:
    - In the **Explorer** view in the left Side Bar, right-click any file under the application name `tanzu-java-web-app` and click `Tanzu: Live Update Start`.
    - Right-click the `tanzu-java-web-app` in the **TANZU WORKLOADS** section of the **Explorer** view and click `Tanzu: Live Update Start`.
    - From the Command Palette, ⇧⌘P on Mac and Ctrl+Shift+P on Windows, type in and select `Tanzu: Live Update Start`.

    You can view output from Tanzu Application Platform indicating that the container is being built and deployed.

    The status of Live Update is reflected in the **TANZU WORKLOADS** view under the `tanzu-java-web-app` workload entry. You can also see `Live Update starting...` in the status bar at the bottom right. Live update can take up to three minutes while the workload deploys and the Knative service becomes available.

    Note

    Depending on the type of cluster you use, you might see an error similar to the following:

> `ERROR: Stop! cluster-name might be production. If you're sure you want to deploy there, add: allow_k8s_contexts('cluster-name') to your Tiltfile. Otherwise, switch k8scontexts and restart Tilt.`
>
> Follow the instructions and add the line, `allow_k8s_contexts('cluster-name')` to your `Tiltfile`.

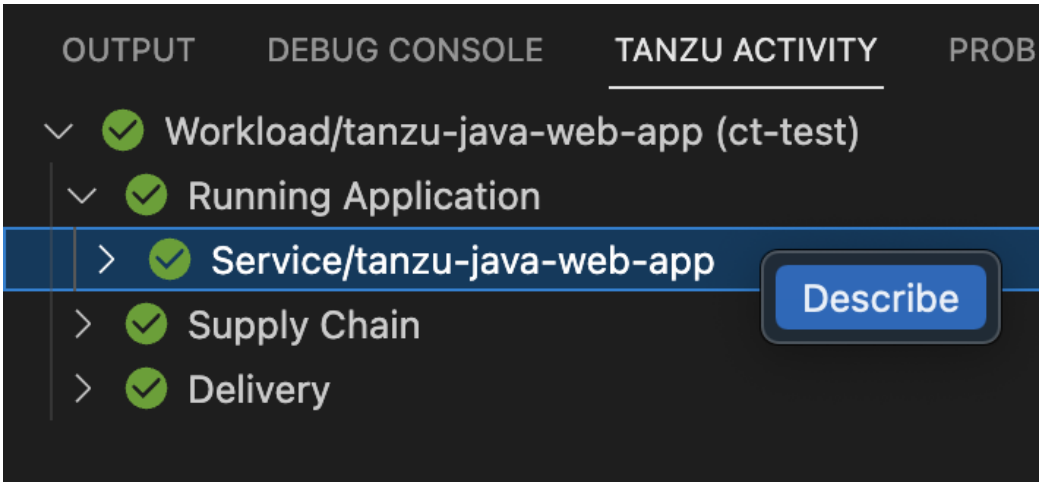
2. When the Live Update status in the **TANZU WORKLOADS** view changes from `Live Update Stopped` to `Live Update Running`, navigate to `http://localhost:8080` in your browser to view your running application.
3. In the IDE, make a change to the source code. For example, in `HelloController.java`, edit the string returned to say `Hello!`, and save.
4. The container is updated when the logs stop streaming. Go to your browser and refresh the page.
5. View the changes to the workload running on the cluster.
6. Either continue making changes, or stop the Live Update process when finished. To stop Live Update, open the Terminal by navigating to **View** > **Terminal**, and click the trash can icon that appears when you place your hover over the **tilt: up - tanzu-java-web-app** process, or select the process and use hot key ⌘+Backspace.

Debug your application

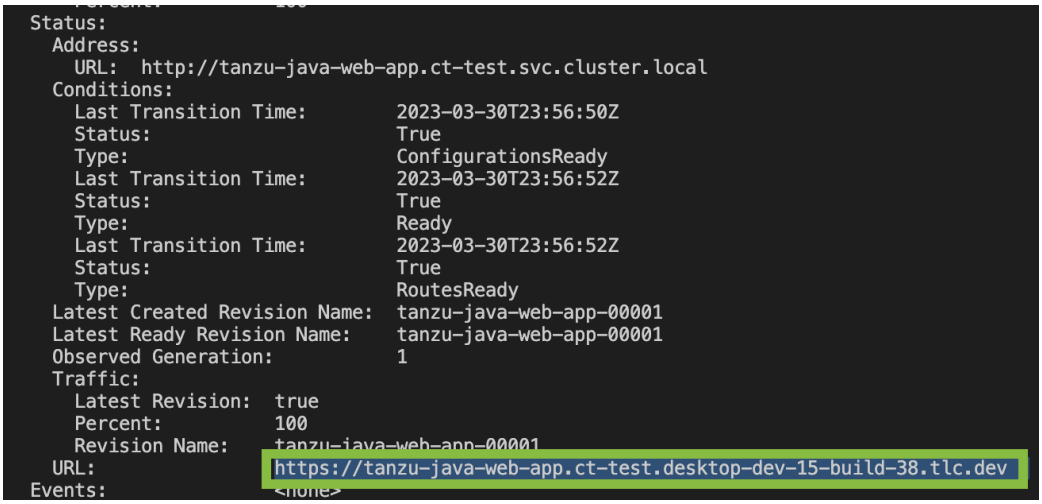
Debug your application in a production-like environment by debugging on your Kubernetes cluster.

To debug the cluster:

1. Set a breakpoint in your code. For example, in `HelloController.java`, set a breakpoint on the line returning text.
2. Apply your application to the cluster.
3. Open the Panel at the bottom of VS Code by clicking **View** > **Appearance** > **Panel**.
4. In the Panel, click the **TANZU ACTIVITY** tab.
5. In the **TANZU ACTIVITY** tab, go to **Workload/tanzu-java-web-app** > **Running Application** > **Service/tanzu-java-web-app**.
6. Right-click the **Pod...** entry and select **Describe**.



7. In resulting output, copy the value after **Status** > **URL:** that begins with `https://tanzu-java-web-app...`. Make sure you copy the value from **Status** > **URL:** and *not* the value under **Status** > **Address** > **URL**.



8. Open your web browser and paste the URL you copied to access your workload.
9. Begin debugging the workload on the cluster by doing one of the following:
    - In the **Explorer** view in the left Side Bar, right-click any file under the application name `tanzu-java-web-app` and click **Tanzu: Java Debug Start**.
    - Alternatively, right-click the `tanzu-java-web-app` in the **TANZU WORKLOADS** view and click **Tanzu: Java Debug Start**.
10. In a few moments, debugging is enabled on the workload. The **Deploy and Connect** task completes and the debug actions are made available to you in the debug overlay, indicating that the debugger has attached.

    The **TANZU WORKLOADS** view shows **Debug Running** under the `tanzu-java-web-app` workload.
11. In your web browser, reload your workload. VS Code opens to show your breakpoint.
12. You can now continue the program, or stop debugging, using the debug controls overlay.

Monitor your running application

Inspect the runtime characteristics of your running application using the Application Live View UI to monitor:

- Resource consumption
- Java Virtual Machine (JVM) status
- Incoming traffic
- Change log level

You can also troubleshoot environment variables and fine-tune the running application.

Use the following steps to diagnose Spring Boot-based applications by using Application Live View:

1. Confirm that the Application Live View components are installed. For instructions, see Install Application Live View.
2. Access the Application Live View UI plug-in in Tanzu Application Platform GUI. For instructions, see Entry point to Application Live View plug-in.
3. Select your running application to view the diagnostic options and inside the application. For more information, see Application Live View features.

Delete your application from the cluster

You can use the delete action to remove your application from the cluster by doing one of the following:

- In the **Explorer** view in the left Side Bar, right-click any file under the application name `tanzu-java-web-app` and click **Tanzu: Delete Workload** to delete the workload from the cluster.
- Alternatively, right-click the `tanzu-java-web-app` in the **TANZU WORKLOADS** view and click **Tanzu: Delete Workload**.

Next steps

- Consume services on Tanzu Application Platform

Claim services on Tanzu Application Platform

This topic for application operators guides you through claiming a service instance and therefore making credentials available to workloads within your namespace. The topic uses RabbitMQ as an example, but the process is the same regardless of the service you want to consume.

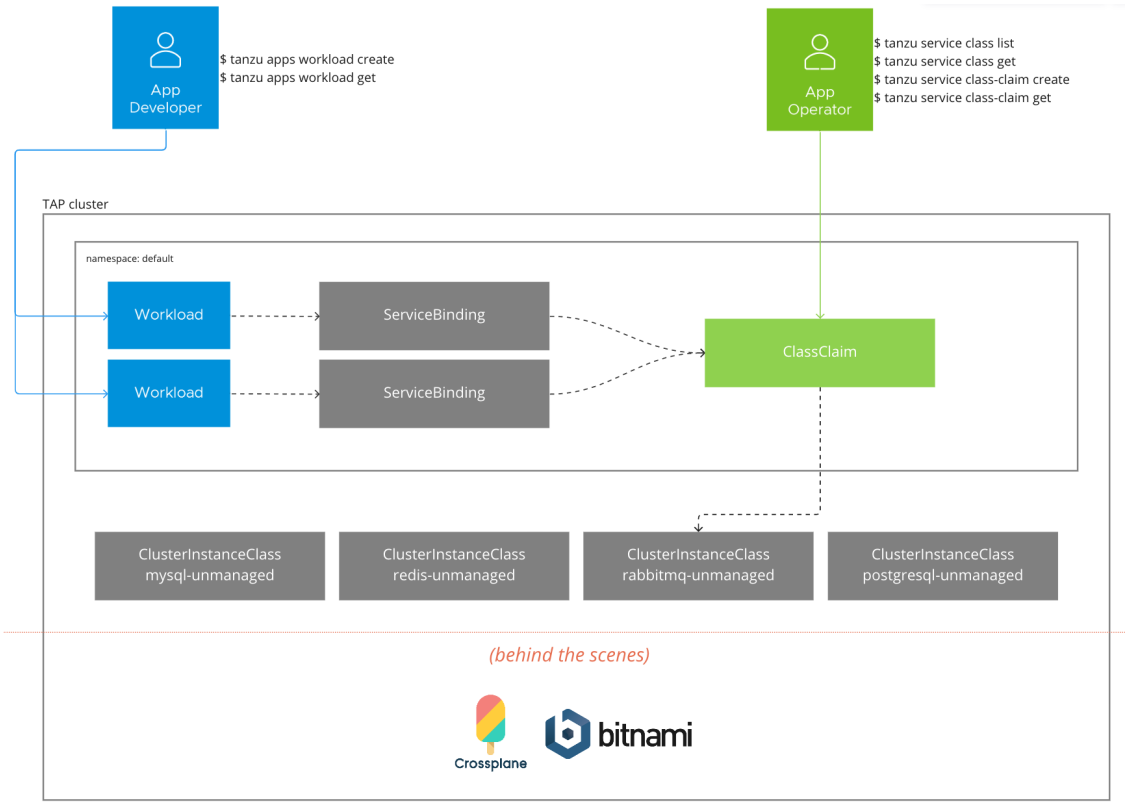
You will use the `tanzu service` CLI plug-in and will learn about classes, claims, and bindings.

What you will do

- Discover the range of services available to you
- Create a claim for an instance of one of the services

Overview

The following diagram depicts a summary of what this tutorial covers.



Bear the following observations in mind as you work through this guide:

1. There are a set of four service classes preinstalled on the cluster.
2. Service operators do not need to configure or setup these four services.
3. The life cycle of a service binding is implicitly tied to the life cycle of a workload, and is managed by the application developer.
4. The life cycles of claims are explicitly managed by the application operator.
5. The diagram and tutorial in this guide are predominantly focused on the application operator, therefore the inner workings of how service instances are provisioned are not in the diagram and are labeled as “behind the scenes”.

Prerequisites

Before following this tutorial, an application operator must:

1. Have access to a cluster with Tanzu Application Platform installed.
2. Have the Tanzu CLI and the corresponding plug-ins.
3. Have access to the `default` namespace which has been set up to use installed packages. For more information, see Set up developer namespaces to use your installed packages.

Discover available services

This section covers using `tanzu service class list` and `tanzu service class get` to find information about the classes of services.

- To discover the range of available services, run the `tanzu service class list` command:

  ```
  tanzu service class list
  ```

  Expected output:

  ```
    NAME                  DESCRIPTION
    mysql-unmanaged       MySQL by Bitnami
    postgresql-unmanaged  PostgreSQL by Bitnami
    rabbitmq-unmanaged    RabbitMQ by Bitnami
    redis-unmanaged       Redis by Bitnami
  ```

  The output lists four classes that cover a range of services: MySQL, PostgreSQL, RabbitMQ and Redis. This is the default set of services that come preconfigured with Tanzu Application Platform. They are backed by Bitnami Helm charts that run on the Tanzu Application Platform cluster. You can consider these to be unmanaged services with no guarantees of service provided.

- To see more detailed information for a class, run the `tanzu service class get` command:

  ```
  tanzu service class get rabbitmq-unmanaged
  ```

  Expected output:

  ```
    NAME:           rabbitmq-unmanaged
    DESCRIPTION:    RabbitMQ by Bitnami
    READY:          true

    PARAMETERS:
      KEY        DESCRIPTION
  TYPE     DEFAULT  REQUIRED
      replicas   The desired number of replicas forming the cluster
  integer  1        false
  ```

```
    storageGB  The desired storage capacity of a single replica, in Gigabytes.
integer  1        false
```

The `PARAMETERS` section is of particular interest because it lists the range of configuration options available to you when creating a claim for the given class.

Create a claim for a service instance

This section covers using `tanzu service class-claim create` to create a claim for an instance of a class and using `tanzu service class-claim get` to get detailed information about the status of the claim.

- To create a claim for an instance of a class, run the `tanzu service class-claim create` command:

  ```
  tanzu service class-claim create rabbitmq-1 --class rabbitmq-unmanaged --parame
  ter storageGB=3
  ```

  In this example, you create a claim for the `rabbitmq-unmanaged` class and pass a parameter to the command to set the storage capacity of the resulting instance to 3 Gigabytes, rather than using the default 1 Gigabyte.

  Expected output:

  ```
    Creating claim 'rabbitmq-1' in namespace 'default'.
  ```

- To get detailed information about the claim, run the `tanzu service class-claim get` command:

  ```
  tanzu service class-claim get rabbitmq-1
  ```

  Expected output:

  ```
    Name: rabbitmq-1
    Namespace: default
    Claim Reference: services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rabbitmq-
  1
    Class Reference:
      Name: rabbitmq-unmanaged
    Parameters:
      storageGB: 3
    Status:
      Ready: True
      Claimed Resource:
        Name: b5982046-a1e9-40cf-8282-00fe67a2f868
        Namespace: default
        Group:
        Version: v1
        Kind: Secret
  ```

  It might take a moment or two for the claim to report `Ready: True`.

In the background, the creation of the claim triggers the on-demand creation of a Helm release of the Bitnami RabbitMQ Helm chart. Credentials and connectivity information required to connect to the RabbitMQ cluster are formatted according to the Service Binding Specification for Kubernetes and stored in a `Secret` in your namespace.

As an application operator you don't need to know what's happening in the background. Tanzu Application Platform promotes a strong separation of concerns between service operators, who are responsible for managing service instances for the platform, and application operators, who want to

use those service instances with their application workloads. The class and claims abstractions enable that separation of concerns. Application operators create claims and service operators help to fulfil them.

Now that you have a claim for a RabbitMQ service instance, you can now follow instructions to Consume services on Tanzu Application Platform.

Learn more

To learn more about working with services on Tanzu Application Platform, see the Services Toolkit component documentation:

- Tutorials
- How-to guides
- Explanations
- Reference material

Next steps

Now that you completed the Getting started guides, learn about:

- Multicluster Tanzu Application Platform

Consume services on Tanzu Application Platform

This topic for application developers guides you through deploying two application workloads and configuring them to communicate using a service instance. The topic uses RabbitMQ as an example, but the process is the same regardless of the service you want to consume.

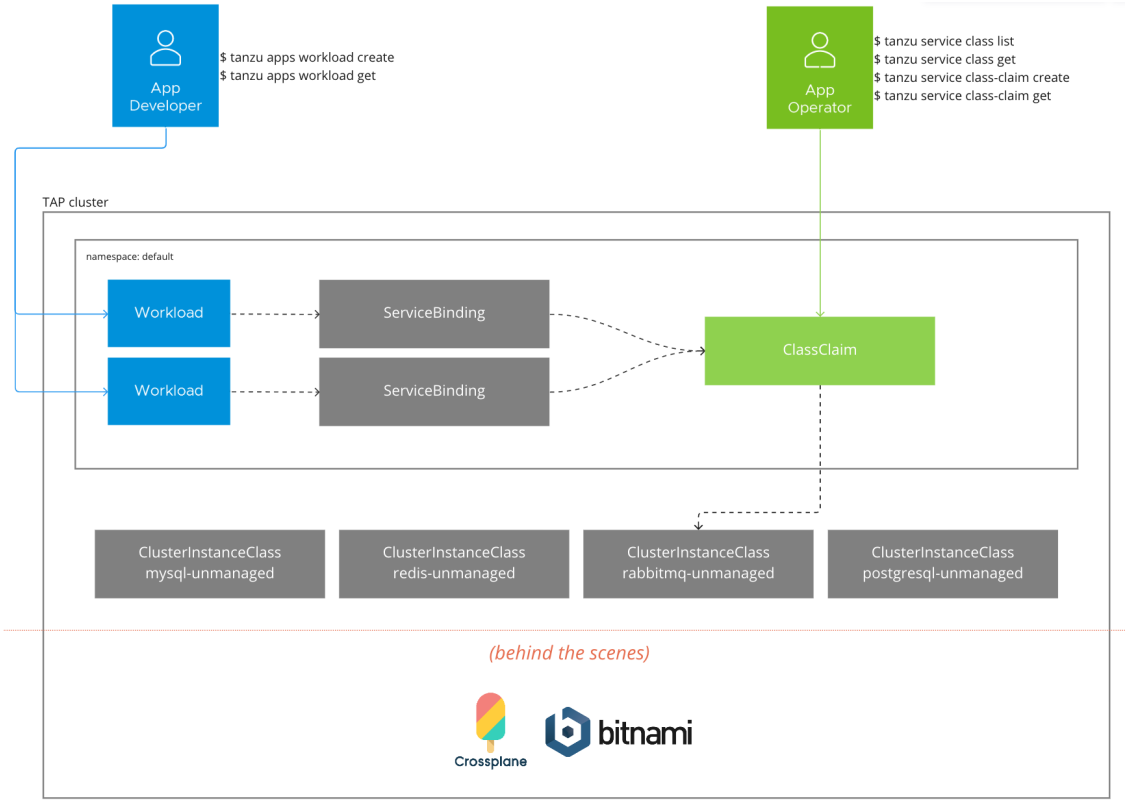
You will use the Tanzu Service CLI plug-in and will learn about classes, claims, and bindings.

What you will do

- Discover existing claims on service instances within your namespace
- Create two application workloads and bind them to an existing claim so that the workloads use the service instance.

Overview

The following diagram depicts a summary of what this tutorial covers.



Bear the following observations in mind as you work through this guide:

1. There are a set of four service classes preinstalled on the cluster.
2. Service operators do not need to configure or setup these four services.
3. The life cycle of a service binding is implicitly tied to the life cycle of a workload, and is managed by the application developer.
4. The life cycles of claims are explicitly managed by the application operator.
5. The diagram and tutorial in this guide are predominantly focused on the application operator and developer user roles, as such the inner workings of how service instances are provisioned are not in the diagram and are labeled as “behind the scenes”.

Prerequisites

Before following this tutorial, an application developer must:

1. Have access to a cluster with Tanzu Application Platform installed.
2. Have the Tanzu CLI and the corresponding plug-ins.
3. Have access to the `default` namespace which has been set up to use installed packages. For more information, see Set up developer namespaces to use your installed packages.
4. Have a Tanzu Application Platform cluster that can pull source code from GitHub.

Discovering existing claims

This section covers using `tanzu service class-claim list` and `tanzu service class-class get` to discover existing claims within your namespace and obtaining information needed to bind your workload to them.

1. To get the list of claims within your namespace, run:

```
tanzu service class-claim list
```

Expected output:

```
NAME        CLASS                READY  REASON
rabbitmq-1  rabbitmq-unmanaged  True   Ready
```

2. To get detailed information about the claim, run:

```
tanzu service class-claim get rabbitmq-1
```

Expected output:

```
  Name: rabbitmq-1
  Namespace: default
  Claim Reference: services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rabbitmq-
1
  Class Reference:
    Name: rabbitmq-unmanaged
  Parameters:
    storageGB: 3
  Status:
    Ready: True
    Claimed Resource:
      Name: b5982046-a1e9-40cf-8282-00fe67a2f868
      Namespace: default
      Group:
      Version: v1
      Kind: Secret
```

Binding application workloads to the service instance

This section covers using `tanzu apps workload create` with the `--service-ref` flag to create workloads and to bind them to the service instance through the claim.

In Tanzu Application Platform, service bindings are created when you create application workloads using the `--service-ref` flag of the `tanzu apps workload create` command.

To create an application workload:

1. Review the output of the `tanzu service class-claim get` command you ran in Discovering existing claims earlier, and note the value of the `Claim Reference`. This is the value to pass to `--service-ref` when creating the application workloads.
2. Create the application workload by running:

```
tanzu apps workload create spring-sensors-consumer-web \
  --git-repo https://github.com/tanzu-end-to-end/spring-sensors \
  --git-branch rabbit \
  --type web \
  --label app.kubernetes.io/part-of=spring-sensors \
  --annotation autoscaling.knative.dev/minScale=1 \
  --service-ref="rmq=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rabbitm
q-1"

tanzu apps workload create \
  spring-sensors-producer \
  --git-repo https://github.com/tanzu-end-to-end/spring-sensors-sensor \
  --git-branch main \
  --type web \
  --label app.kubernetes.io/part-of=spring-sensors \
  --annotation autoscaling.knative.dev/minScale=1 \
```

```
  --service-ref="rmq=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:rabbitm
q-1"
```

3. After the workloads are ready, visit the URL of the `spring-sensors-consumer-web` app. Confirm that sensor data is passing from the `spring-sensors-producer` workload to the `spring-sensors-consumer-web` workload using the `RabbitmqCluster` service instance.

Learn more

To learn more about working with services on Tanzu Application Platform, see the Services Toolkit component documentation:

- Tutorials
- How-to guides
- Concepts
- Reference material

Next steps

Now that you completed the Getting started guides, learn about:

- Multicluster Tanzu Application Platform

Deploy an air-gapped workload on Tanzu Application Platform

This topic for developers guides you through deploying your first workload on Tanzu Application Platform (commonly known as TAP) in an air-gapped environment.

For information about installing Tanzu Application Platform in an air-gapped environment, see Install Tanzu Application Platform in an air-gapped environment.

What you will do

- Create a workload from Git.
- Create a basic supply chain workload.
- Create a testing supply chain workload.
- Create a testing scanning supply chain workload.

Prerequisites

Before you begin, a Platform operator must configure the air-gapped environment using Namespace Provisioner. For instructions, see Work with Git repositories in air-gapped environments with Namespace Provisioner.

Create a workload from Git

To create a workload from Git through HTTPS, follow these steps:

1. (Optional) To pass in login credentials for a Git repository with the certificate authority (CA) certificate, create a file called `git-credentials.yaml`. For example:

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: git-ca
  # namespace: default
type: Opaque
stringData:
  username: USERNAME
  password: PASSWORD
  caFile: |
    CADATA
```

Where:

- `USERNAME` is the user name.
- `PASSWORD` is the password.
- `CADATA` is the PEM-encoded CA certificate for the Git repository.

2. To pass in a custom `settings.xml` for Java, create a file called `settings-xml.yaml`. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: settings-xml
type: service.binding/maven
stringData:
  type: maven
  provider: sample
  settings.xml: |
    <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://maven.
apache.org/xsd/settings-1.0.0.xsd">
        <mirrors>
            <mirror>
                <id>reposilite</id>
                <name>Tanzu seal Internal Repo</name>
                <url>https://reposilite.tap-trust.cf-app.com/releases</url>
                <mirrorOf>*</mirrorOf>
            </mirror>
        </mirrors>
        <servers>
            <server>
                <id>reposilite</id>
                <username>USERNAME</username>
                <password>PASSWORD</password>
            </server>
        </servers>
    </settings>
```

3. Apply the file:

```
kubectl create -f settings-xml.yaml -n DEVELOPER-NAMESPACE
```

Create a basic supply chain workload

Next, create your basic supply chain workload.

To pass the CA certificate in when you create the workload, run:

```
tanzu apps workload create APP-NAME --git-repo  https://GITREPO --git-branch BRANCH --
type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml buildService
```

```
Bindings='[{"name": "settings-xml", "kind": "Secret"}]' --param "gitops_ssh_secret=git
-ca"
```

Create a testing supply chain workload

For instructions about creating a workload with the testing supply chain, see Install OOTB Supply Chain with Testing.

To add the Tekton supply chain to the cluster, apply the following YAML to the cluster:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test     # (!) required
spec:
  params:
    - name: source-url                       # (!) required
    - name: source-revision                  # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: MY-REGISTRY/gradle
            script: |-
              cd `mktemp -d`
```

Where `MY-REGISTRY` is your container image registry. Relocate all the images given in the pipeline YAML to your private container registry.

Create the workload by running:

```
tanzu apps workload create APP-NAME --git-repo  https://GITURL --git-branch BRANCH --t
ype web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label apps.
tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]'
```

To instead pass the CA certificate when you create the workload, run:

```
tanzu apps workload create APP-NAME --git-repo  https://GITREPO --git-branch BRANCH --
type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label app
s.tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]' --param "gitops_ssh_secret=git-ca"
```

Create a testing scanning supply chain workload

For instructions about creating a workload with the testing and scanning supply chain, see Install OOTB Supply Chain with Testing and Scanning.

In addition to the prerequisites given at Prerequisites, follow Using Grype in offline and air-gapped environments before workload creation.

Create workload by running:

```
tanzu apps workload create APP-NAME --git-repo  https://GITURL --git-branch BRANCH --t
ype web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label apps.
tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]'
```

To instead pass the CA certificate when you create the workload, run:

```
tanzu apps workload create APP-NAME --git-repo  https://GITREPO --git-branch BRANCH --
type web --label app.kubernetes.io/part-of=CATALOGNAME --yes --param-yaml --label app
s.tanzu.vmware.com/has-tests=true buildServiceBindings='[{"name": "settings-xml", "kin
d": "Secret"}]' --param "gitops_ssh_secret=git-ca"
```

Deploy Spring Cloud applications to Tanzu Application Platform

This sub-section tells you how to run Spring applications that rely on various Spring Cloud services as workloads on Tanzu Application Platform (commonly known as TAP).

In this sub-section:

- Deploy Spring Cloud Config applications
- Deploy Spring Cloud DiscoveryClient applications
- Use Spring Cloud Gateway for Kubernetes

Deploy Spring Cloud applications to Tanzu Application Platform

This sub-section tells you how to run Spring applications that rely on various Spring Cloud services as workloads on Tanzu Application Platform (commonly known as TAP).

In this sub-section:

- Deploy Spring Cloud Config applications
- Deploy Spring Cloud DiscoveryClient applications
- Use Spring Cloud Gateway for Kubernetes

Deploy Spring Cloud Config applications to Tanzu Application Platform

This topic tells you how to run Spring applications that depend on Spring Cloud Config Server as workloads on Tanzu Application Platform (commonly known as TAP).

Identify Spring Cloud Config applications

The Spring Cloud Config project is used within many common configuration services for Spring applications, including the following:

- The Config Server in the managed service tile Spring Cloud Services for VMware Tanzu that is supported by VMware Tanzu Application Service for VMs.
- Application Configuration Service for Tanzu in Azure Spring Apps. For more information about Azure Spring Apps, see the Microsoft Azure documentation.

Spring applications that use these configuration services often include a client dependency that interacts with the Spring Cloud Config Server:

- Applications that use the Spring Cloud Services Config Server on Tanzu Application Service typically include the `spring-cloud-services-starter-config-client` dependency from the `io.pivotal.spring.cloud` group. For more information, see the Config Server in the Spring Cloud Services documentation.
- Applications that use the open-source Spring Cloud Config Server typically include the `spring-cloud-starter-config` dependency from the `org.springframework.cloud` group. For more information, see the Spring Cloud Config documentation.

Prerequisites

Before you can deploy Spring Cloud Config applications, you must Install Application Configuration Service for VMware Tanzu.

The Application Configuration Service for VMware Tanzu component in Tanzu Application Platform distributes configuration information to applications through Kubernetes Secrets that contain Spring properties.

Configure workloads

For instructions for how to run existing Spring applications that rely on the Spring Cloud Config Server as workloads in Tanzu Application Platform, see Configuring Workloads in Tanzu Application Platform using Application Configuration Service in the Application Configuration Service for VMware Tanzu documentation.

Deploy Spring Cloud DiscoveryClient applications to Tanzu Application Platform

This topic tells you how to run Spring applications that use the Spring Cloud DiscoveryClient as workloads on Tanzu Application Platform (commonly known as TAP).

Identify Spring Cloud DiscoveryClient applications

The Spring Cloud DiscoveryClient abstraction underlies several common libraries and services for Spring applications to register themselves as services for other applications and to look up connection details of registered applications. These services include the following:

- The Service Registry in the managed service tile Spring Cloud Services for VMware Tanzu supported by VMware Tanzu Application Service for VMs.
- The Tanzu Service Registry in Azure Spring Apps. For more information about Azure Spring Apps, see the Microsoft Azure documentation.
- The Spring Cloud Netflix project, which includes the Eureka client library and the Eureka server.

Spring applications that use these discovery services include a client dependency that implements the Spring Cloud DiscoveryClient:

- Applications that use the Spring Cloud Services Service Registry on Tanzu Application Service typically include the `spring-cloud-services-starter-service-registry` dependency from the `io.pivotal.spring.cloud` group. For more information, see Service Registry in the Spring Cloud Services documentation.

- Applications that use the Tanzu Service Registry in Azure Spring Apps or that use the Spring Cloud Netflix libraries typically include the `spring-cloud-starter-netflix-eureka-client` dependency from the `org.springframework.cloud` group. For more information about how to use the Tanzu Service Registry, see the Microsoft Azure documentation. For more information about how to include Eureka Client, see the Spring documentation.

Each of these client dependencies includes the Spring Cloud SimpleDiscoveryClient from the Spring Cloud Commons project as a base dependency. The approach in this topic uses this common dependency to configure service resolution for client applications.

Prerequisites

Before you can continue with the example in this topic, you must Install Application Configuration Service for VMware Tanzu.

In this example, the Application Configuration Service for VMware Tanzu component in Tanzu Application Platform distributes service discovery information to client applications as Spring properties.

Example: The Greeting application

The following sections show how to run the Greeting sample application as a pair of workloads on Tanzu Application Platform.

Create a properties file in your configuration repository

In a Git repository that is reachable from your Run cluster, create a `greeter-dev.yaml` file as follows:

```
eureka:
  client:
    # this disables the Eureka Spring Cloud discovery client
    enabled: false
spring:
  cloud:
    discovery:
      client:
        simple:
          instances:
            greeter-messages:
            - uri: http://greeter-messages.my-apps.svc.cluster.local
```

The values under `spring.cloud.discovery.client.simple.instances` list all the services that your application requires. The example `greeter-dev.yaml` file shows how to connect to another workload running on the same cluster.

In the example in Create application workload resources, the `greeter-messages` microservice is deployed as a workload of type `web`, so the discovery client configuration must use the fully qualified domain name for the service within the Kubernetes cluster. If you instead choose to run the `greeter-messages` microservice as a workload of type `server`, this address still works, but the `greeter` microservice can also connect using the shorter URI `http://greeter-messages`.

Create Application Configuration Service resources

On your Run cluster, create the `ConfigurationSource` and `ConfigurationSlice` resources that tell Application Configuration Service (ACS) how to fetch the discovery configuration from the Git repository you are using.

The following example uses a public repository and no encryption. For more information about how to connect to private repositories, encrypt configuration, and load properties in other formats, see

the ACS documentation.

```
---
apiVersion: "config.apps.tanzu.vmware.com/v1alpha4"
kind: ConfigurationSource
metadata:
  name: greeter-config-source
  namespace: my-apps
spec:
  backends:
    - type: git
      uri: https://github.com/your-org/your-config-repo
---
apiVersion: config.apps.tanzu.vmware.com/v1alpha4
kind: ConfigurationSlice
metadata:
  name: greeter-config
  namespace: my-apps
spec:
  configurationSource: greeter-config-source
  content:
  - greeter/dev
  secretStrategy: applicationProperties
  interval: 10m
```

A Kubernetes secret is created in the `my-apps` namespace with a name starting with `greeter-config-`.

Create application workload resources

The `ConfigurationSlice` object you created in the previous section is a Provisioned Service. You can use a `ResourceClaim` to claim it within the `my-apps` namespace. You then supply the resource claim in the `serviceClaims` list in the `Workload` object to provide the configuration inside the runtime environment of the workload.

The `SPRING_CONFIG_IMPORT` variable passes this configuration to Spring. If your application already uses that variable to apply other Spring configuration, use the `SPRING_CONFIG_ADDITIONAL_LOCATION` variable instead.

In the following example, one workload is created for the `greeter-messages` microservice, and a second workload is created for the greeter microservice. Both apps bind to the `ConfigurationSlice` to add Spring configuration:

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: greeter-messages
  namespace: my-apps
  labels:
    apps.tanzu.vmware.com/workload-type: web
    apps.tanzu.vmware.com/has-tests: "true"
    app.kubernetes.io/part-of: greeter
spec:
  build:
    env:
    - name: BP_JVM_VERSION
      value: "17"
    # this tells the Gradle buildpack which module to build
    - name: BP_GRADLE_BUILT_MODULE
      value: "greeter-messages"
  env:
  # the Greeting app enables basic authentication unless the
```

```
  # development profile is used
  - name: SPRING_PROFILES_ACTIVE
    value: "development"
  - name: SPRING_CONFIG_IMPORT
    value: "${SERVICE_BINDING_ROOT}/spring-properties/"
  serviceClaims:
  - name: spring-properties
    ref:
      apiVersion: services.apps.tanzu.vmware.com/v1alpha1
      kind: ResourceClaim
      name: greeter-config-claim
  source:
    git:
      url: https://github.com/spring-cloud-services-samples/greeting
      ref:
        branch: main
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: greeter
  namespace: my-apps
  labels:
    apps.tanzu.vmware.com/workload-type: web
    apps.tanzu.vmware.com/has-tests: "true"
    app.kubernetes.io/part-of: greeter
spec:
  build:
    env:
    - name: BP_JVM_VERSION
      value: "17"
    - name: BP_GRADLE_BUILT_MODULE
      value: "greeter"
  env:
  - name: SPRING_PROFILES_ACTIVE
    value: "development"
  - name: SPRING_CONFIG_IMPORT
    value: "${SERVICE_BINDING_ROOT}/spring-properties/"
  serviceClaims:
  - name: spring-properties
    ref:
      apiVersion: services.apps.tanzu.vmware.com/v1alpha1
      kind: ResourceClaim
      name: greeter-config-claim
  source:
    git:
      url: https://github.com/spring-cloud-services-samples/greeting
      ref:
        branch: main
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaim
metadata:
  name: greeter-config-claim
  namespace: my-apps
spec:
  ref:
    apiVersion: config.apps.tanzu.vmware.com/v1alpha4
    kind: ConfigurationSlice
    name: greeter-config
```

The greeter application builds, starts up, and finds the `greeter-messages` URI using the SimpleDiscoveryClient.

Using Spring Cloud Gateway for Kubernetes with Tanzu Application Platform

This topic tells you how to use Spring Cloud Gateway for Kubernetes as an API gateway for workloads running on Tanzu Application Platform (commonly known as TAP).

Spring Cloud Gateway is a popular project library for creating an API Gateway that is built on top of the Spring ecosystem. The open source library is a foundational component of VMware Spring Cloud Gateway for Kubernetes and Spring Cloud Gateway for VMware Tanzu Application Service commercial offerings with commercial-only capabilities and platform-integrated operator experiences. You can use the open source and commercial offerings as a reverse proxy with extra API Gateway functions to handle request and response to upstream application services.

Spring Cloud Gateway for Kubernetes is included with Tanzu Application Platform v1.5 and later. You can migrate upstream applications that expose API routes on Spring Cloud Gateway from Tanzu Application Service and custom open source implementations to Tanzu Application Platform. For how to do so, see the VMware Spring Cloud Gateway for Kubernetes documentation.

Create a new application accelerator

This topic guides you through creating an accelerator and registering it in a Tanzu Application Platform (commonly known as TAP) instance.

Tanzu Application Platform offers a selection of built-in accelerators to streamline your development process. However, if these accelerators don't meet your needs, you can create a new accelerator. By creating an accelerator, you can ensure that your technology stacks and organizational best practices are adhered to.

Note

This guide follows a quick start format. See the Application Accelerator documentation for advanced features.

What you will do

- Create a new accelerator project that contains an `accelerator.yaml` file and `README.md` file.
- Configure the `accelerator.yaml` file to alter the project's `README.md`.
- Test your accelerator locally using the Tanzu CLI `generate-from-local` command.
- Create a new Git repository for the project and push the project to it.
- Register the accelerator in a Tanzu Application Platform instance.
- Verify project generation with the new accelerator by using Tanzu Application Platform GUI.

Set up Visual Studio Code

1. To simplify accelerator authoring, code assist capabilities are available. To install the extension, navigate to the Marketplace page for the YAML plug-in and click **Install**.

   Note

   Code assist for authoring accelerators is also available in the IntelliJ IDE. You can enable this by selecting **Application Accelerator** in the schema

mapping drop-down menu. For more information about how to enable this, see the IntelliJ Using schemas from JSON Schema Store documentation.

2. After you install the plug-in, editing files entitled `accelerator.yaml` automatically uses the code assist capabilities.

Create a simple project

To create your project, follow these instructions to set up the project directory, prepare the `README.md` and `accelerator.yaml`, and test your accelerator.

Set up the project directory

1. Create a new directory for the project named `myProject` and change to the newly created directory.

```
mkdir myProject
cd myProject
```

2. Create two new files in the `myProject` directory named `README.md` and `accelerator.yaml`.

```
touch README.MD accelerator.yaml
```

Prepare the `README.md` and `accelerator.yaml`

The following instructions require using Visual Studio Code to edit the files.

1. Using Visual Studio Code, open the `README.md`, copy and paste the following code block into it, and save the file. `CONFIGURABLE_PARAMETER_#` is targeted to be transformed during project generation in the upcoming `accelerator.yaml` definition.

```
## Tanzu Application Accelerator Sample Project

This is some very important placeholder text that should describe what this pro
ject can do and how to use it.

Here are some configurable parameters:

* CONFIGURABLE_PARAMETER_1
* CONFIGURABLE_PARAMETER_2
```

2. Open `accelerator.yaml` and begin populating the file section using the snippet below. This section contains important information, such as the accelerator's display name, description, tags, and more.

   For all possible parameters available in this section, see Creating accelerator.yaml.

```
accelerator:
  displayName: Simple Accelerator
  description: Contains just a README
  iconUrl: https://blogs.vmware.com/wp-content/uploads/2022/02/tap.png
  tags:
    - simple
    - getting-started
```

3. Add the configuration parameters using the following code snippet. This configures what parameters are displayed in the accelerator form during project creation.

In this example snippet, the field `firstConfigurableParameter` takes in text the user provides. The `secondConfigurableParameter` does the same, except it is only displayed if the user checks `secondConfigurableParameterCheckbox` because of the `dependsOn` parameter.

For more information about possible options, see Creating accelerator.yaml.

```
# Place this after the 'tags' section from the previous step
  options:
    - name: firstConfigurableParameter
      inputType: text
      label: The text used to replace the first placeholder text in the README.
md. Converted to lowercase.
      defaultValue: Configurable Parameter 1
      required: true
    - name: secondConfigurableParameterCheckbox
      inputType: checkbox
      dataType: boolean
      label: Enable to configure the second configurable parameter, otherwise u
se the default value.
    - name: secondConfigurableParameter
      inputType: text
      label: The text used to replace the second placeholder text in the READM
E.md. Converted to lowercase.
      defaultValue: Configurable Parameter 2
      dependsOn:
        name: secondConfigurableParameterCheckbox
```

4. Add the `engine` configuration by using the following code snippet and save the file.

   The `engine` configuration tells the `accelerator engine` behind the scenes what must be done to the project files during project creation. In this example, this instructs the engine to replace `CONFIGURABLE_PARAMETER_1` and, if the check box is checked, `CONFIGURABLE_PARAMETER_2` with the parameters that the user passes in during project creation.

   This also leverages Spring Expression Language (SpEL) syntax to convert the text input to all lowercase.

   For more information about the possible parameters for use within the `engine` section, see Creating accelerator.yaml.

   ```
   # Place this after the `options` section from the previous step
   engine:
     merge:
       - include: [ "README.md" ]
         chain:
           - type: ReplaceText
             substitutions:
               - text: "CONFIGURABLE_PARAMETER_1"
                 with: "#firstConfigurableParameter.toLowerCase()"
           - condition: "#secondConfigurableParameterCheckbox"
             chain:
             - type: ReplaceText
               substitutions:
                 - text: "CONFIGURABLE_PARAMETER_2"
                   with: "#secondConfigurableParameter.toLowerCase()"
   ```

Test the accelerator

It is important to quickly test and iterate on accelerators as they are being developed to ensure that the resulting project is generated as expected.

1. Using the terminal of your choice with access to the `tanzu` command, run the following command to test the accelerator created earlier.

   This step takes the local `accelerator.yaml` and project files, configures the project using the parameters passed in through the `--options` field, and outputs the project to a specified directory.

   Important

   This step requires that the `TANZU-APPLICATION-ACCELERATOR-URL` endpoint is exposed and accessible. For more information, see Server API connections for operators and developers.

   ```
   tanzu accelerator generate-from-local \
       --accelerator-path simple-accelerator="$(pwd)" `# The path to new accelerat
   or` \
       --server-url TANZU-APPLICATION-ACCELERATOR-URL `# Example: https://accelera
   tor.mytapcluster.myorg.com` \
       --options '{"firstConfigurableParameter": "Parameter 1", "secondConfigurabl
   eParameterCheckbox": true, "secondConfigurableParameter":"Parameter 2"}' \
       -o "${HOME}/simple-accelerator/" `# Change this path to change where the pr
   oject folder gets generated`
   ```

2. After the project is generated, a status message is displayed.

   ```
   generated project simple-accelerator
   ```

3. Navigate to the output directory and verify that the `README.md` is updated based on the `--options` specified in the preceding `generate-from-local` command.

   ```
   ## Tanzu Application Accelerator Sample Project

   This is some very important placeholder text that should describe what this pro
   ject can do and how to use it.

   Here are some configurable parameters:

   - parameter 1
   - parameter 2
   ```

Upload the project to a Git repository

The Application Accelerator system and Tanzu Application Platform GUI depend on an accelerator project residing inside a Git repository. For this example, GitHub is used.

1. Create a new repository in GitHub and ensure that **Visibility** is set to **Public**. Click **Create Repository**.
2. To push your accelerator project (**not** the generated project from `generate-from-local`) to GitHub, follow the instructions that GitHub provides for the *…or create a new repository on the command line* that is shown after clicking **Create Repository**. Instructions can also be found in the GitHub documentation.
3. Verify that the project is pushed to the target repository.

Register the accelerator to the Tanzu Application Platform and verify project generation output

Now that the accelerator is committed to its own repository, you can register the accelerator to Tanzu Application Platform GUI for developers to generate projects from the newly created accelerator.

To do so, use the URL of the Git repository and branch name created earlier and run the following command using the Tanzu CLI to register the accelerator to Tanzu Application Platform GUI.

Note

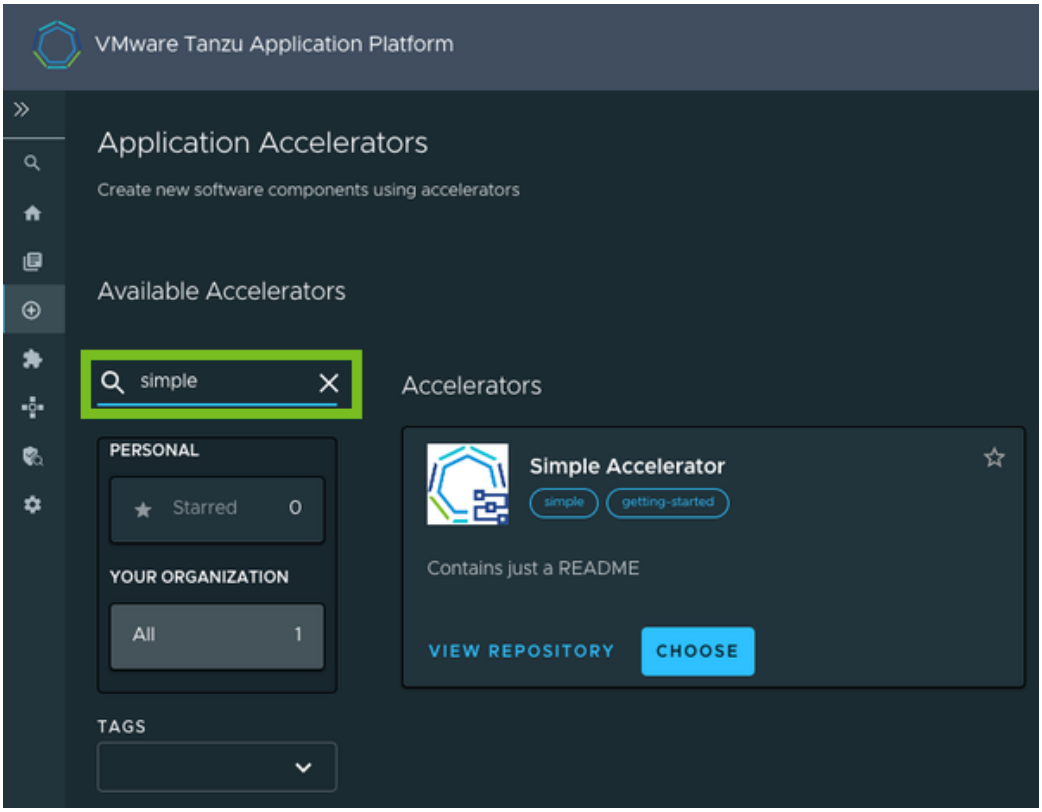
`tanzu accelerator create` works with monorepos as well. Add the `--git-sub-path` parameter with the desired subpath to fetch the accelerator project in that directory. For more information, see tanzu accelerator create.

```
tanzu accelerator create simple-accelerator --git-repository https://github.com/myuser
name/myprojectrepository --git-branch main
```

The accelerator can take time to reconcile. After it has reconciled, it is available for use in Tanzu Application Platform GUI and the Application Accelerator extension for Visual Studio Code.

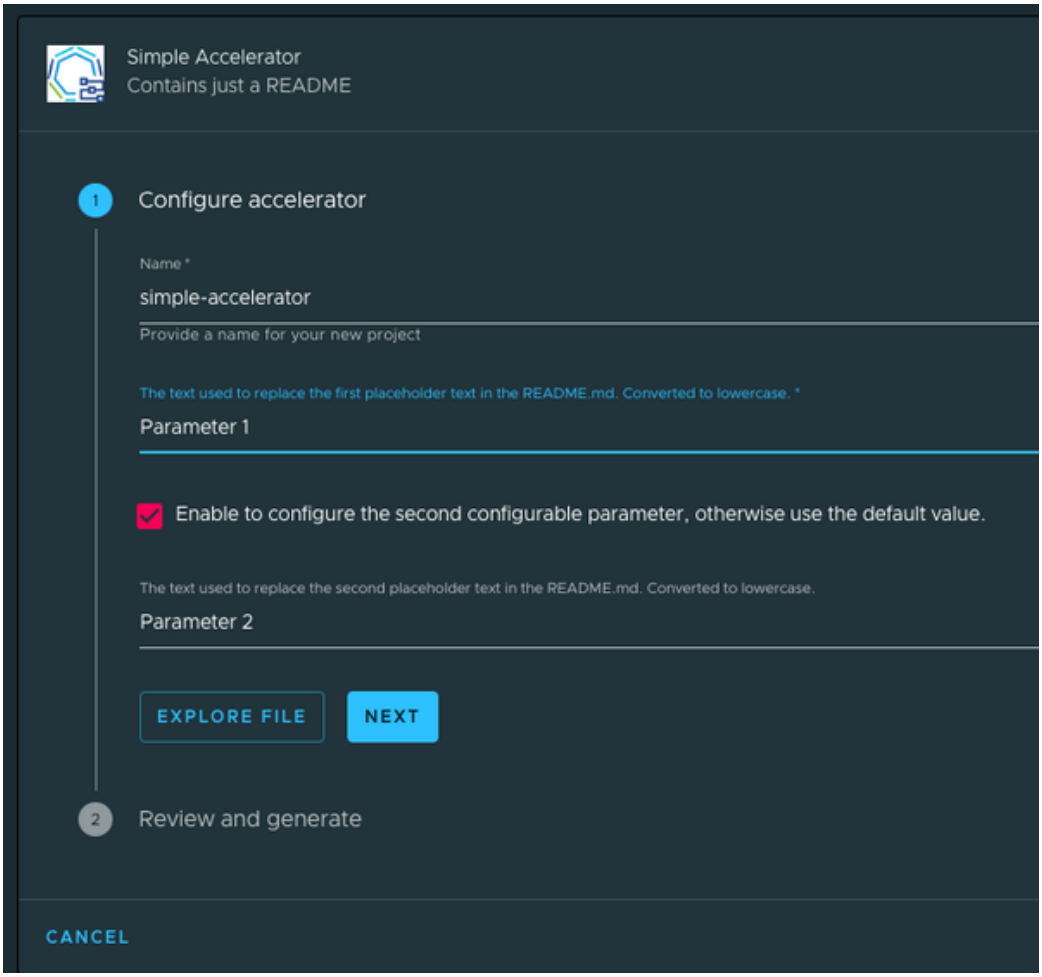
Verify project generation output by using Tanzu Application Platform GUI

1. Navigate to your organization's instance of Tanzu Application Platform GUI.
2. On the left navigation pane, click **Create**.
3. Using the search bar near the left side of the page, search for `simple accelerator`. After you've found it, click **Choose** on the accelerator card.



4. Configure the project by filling in the parameters in the form.

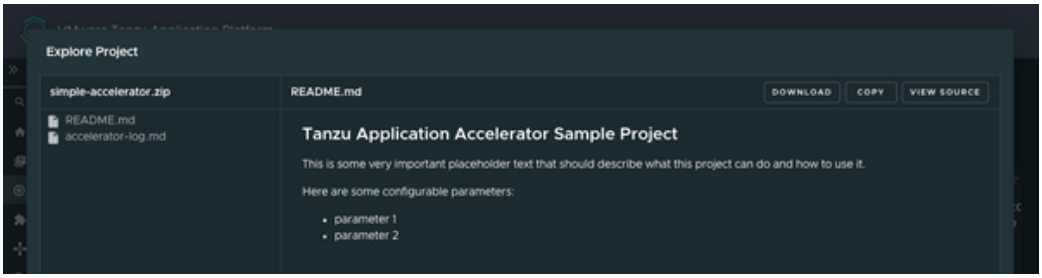
The options you defined in `accelerator.yaml` are now displayed for you to configure. The `secondConfigurableParameter dependsOn secondConfigurableParameterCheckbox` might be hidden depending on whether the check box is selected.



5. After configuration is complete, click **Next**.

> **Note**
>
> Depending on your organization's Tanzu Application Platform configuration, you might be presented with an option to create a Git repository. In this guide, this is skipped and is covered in Deploy an app on Tanzu Application Platform.

6. On the **Review and generate** step, review the parameters and click **Generate Accelerator**.
7. Explore the ZIP file of the configured project and verify that the project is generated with the parameters you provided during configuration.



Learn more about Application Accelerator

- For advanced functionality when creating accelerators, such as accelerator best practices, accelerator fragments, engine transforms, and more, see the Application Accelerator documentation.
- For more information about Application Accelerator configurations, see the Configure Application Accelerator documentation.
- For information about installing the Application Accelerator extension for Visual Studio Code, see the Application Accelerator Visual Studio Code extension documentation.
- For general accelerator troubleshooting, see Troubleshooting Application Accelerator for VMware Tanzu.

Learn about Tanzu Application Platform

The topics in this section explain concepts important to getting started with Tanzu Application Platform (commonly known as TAP).

In this section:

- Application Accelerator
- Supply chains on Tanzu Application Platform
- Vulnerability scanning and metadata storage for your supply chain
- Consume services on Tanzu Application Platform

Application accelerators on Tanzu Application Platform

This topic describes the key concepts you need to know about application accelerators on Tanzu Application Platform (commonly known as TAP).

What are application accelerators

Application accelerators are templates that not only codify best practices but also provide important configuration and structures ready and available for use. Developers can create applications and get started with feature development immediately with the help of application accelerators.

Enterprise Architects use Application Accelerator to create application accelerators, which provide developers and admins in their organization with ready-made, enterprise-conforming code and configurations. Accelerators contain complete and runnable application code and deployment configurations. They also contain metadata for altering the code and deployment configurations based on input values provided for specific options defined in the accelerator metadata.

Working with accelerators

The Application Accelerator plug-in for Tanzu Application Platform GUI helps you to discover accelerators and to enter extra information used for processing the files before downloading. As of Tanzu Application Platform v1.2, developers can also discover and work on accelerators right in Visual Studio Code with the Tanzu Application Accelerator for VS Code extension. Developers can use the `list`, `get`, and `generate` commands to use accelerators available in an Application Accelerator server.

Admins use the `create`, `update`, and `delete` commands for managing accelerators in a Kubernetes context. When admins want to use the `get` and `list` commands, they can specify the `--from-context` flag to access accelerators in a Kubernetes context.

Next steps

Apply what you have learned:

Developers:

- Deploy an app on Tanzu Application Platform

Operators:

- Create an application accelerator

Supply chains on Tanzu Application Platform

This topic describes the key concepts you need to know about supply chains and Continuous Integration/Continuous Delivery (CI/CD) on Tanzu Application Platform (commonly known as TAP).

What are supply chains

Supply chains provide a way of codifying all of the steps of your path to production, more commonly known as CI/CD. CI/CD is a method to frequently deliver applications by introducing automation into the stages of application development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment.

CI/CD is the method used by supply chains to deliver applications through automation. Tanzu Application Platform supply chains allow you to use CI/CD and add any other steps necessary for an application to reach production or a different environment, such as staging.

A path to production

A path to production allows you to create a unified access point for all of the tools required for your applications to reach a customer-facing environment. Instead of having four tools that are loosely coupled to each other, a path to production defines all four tools in a single, unified layer of abstraction. The path to production can be automated and repeatable between teams for applications at scale.

Typically tools cannot integrate with one another without scripting or webhooks. Whereas with a path to production, there is a unified automation tool to codify all the interactions between each of the tools. Supply chains that are used to codify the path to production for an organization are configurable. This allows their authors to add all of the steps of the path to production for their applications.

Available supply chains

Tanzu Application Platform provides three out of the box (OOTB) supply chains to work with the Tanzu Application Platform components. They include:

- OOTB Supply Chain Basic (default)
- OOTB Supply Chain with Testing (optional)
- OOTB Supply Chain with Testing+Scanning (optional)

1: OOTB Basic (default)

The default **OOTB Basic** supply chain and its dependencies were installed on your cluster during the Tanzu Application Platform install. The following diagram and table provide a description of the supply chain and dependencies provided with Tanzu Application Platform.

Name	Package Name	Description	Dependencies

Out of the Box Basic (Default - Installed during Installing Part 2)	`ootb-supply-chain-basic.tanzu.vmware.com`	This supply chain monitors a repository that is identified in the developer's `workload.yaml` file. When any new commits are made to the application, the supply chain:
		• Creates a new image.
		• Applies any predefined conventions.
		• Deploys the application to the cluster.
			• Flux/Source Controller
			• Tanzu Build Service
			• Cartographer Conventions
			• Tekton
			• Cloud Native Runtimes
			• If using Service References:
			  ◦ Service Bindings
			  ◦ Services Toolkit

2: OOTB Testing

OOTB Testing supply chain runs a Tekton pipeline within the supply chain. The following diagram and table provide a description of the supply chain and dependencies provided with Tanzu Application Platform.

Name	Package Name	Description	Dependencies
Out of the Box Testing	`ootb-supply-chain-testing.tanzu.vmware.com`	Out of the Box Testing contains all of the same elements as the Source to URL. It allows developers to specify a Tekton pipeline that runs as part of the CI step of the supply chain. • The application tests using the Tekton pipeline. • A new image is created. • Any predefined conventions are applied. • The application is deployed to the cluster.	All of the Source to URL dependencies

3: OOTB Testing+Scanning

OOTB Testing+Scanning supply chain includes integrations for secure scanning tools. The following diagram and table provide a description of the supply chain and dependencies provided with Tanzu Application Platform.

Name	Package Name	Description	Dependencies
Out of the Box Testing and Scanning	`ootb-supply-chain-testing-scanning.tanzu.vmware.com`	Out of the Box Testing and Scanning contains all of the same elements as the Out of the Box Testing supply chain, and it also includes integrations with the secure scanning components of Tanzu Application Platform. • The application is tested using the provided Tekton pipeline. • The application source code is scanned for vulnerabilities. • A new image is created. • The image is scanned for vulnerabilities. • Any predefined conventions are applied. • The application deploys to the cluster.	All of the Source to URL dependencies, and: • The secure scanning components included with Tanzu Application Platform

Next steps

Apply what you have learned:

- Add testing and scanning to your application

Or learn about:

- Vulnerability scanning and metadata storage for your supply chain

Vulnerability scanning, storing, and viewing for your supply chain

This topic describes the vulnerability scanning features you can use with Tanzu Application Platform (commonly known as TAP).

This feature set allows an application operator to introduce source code and image vulnerability scanning, storing, and viewing to their Tanzu Application Platform supply chain. It also allows for the creation of scan-time rules that prevent critical vulnerabilities from flowing to the supply chain unresolved.

Features

Features include:

- Scan source code repositories and images for known common vulnerabilities and exposures (CVEs) before deploying to a cluster.
- Identify CVEs by scanning continuously on each new code commit or each new image built.
- Analyze scan results against user-defined policies by using Open Policy Agent. Create scan policy to prevent vulnerable components from going into production.
- Produce vulnerability scan results and post them to the SCST - Store where they can be queried.
- Query the store for such use cases as:
    - What images and packages are affected by a specific vulnerability?
    - What source code repositories are affected by a specific vulnerability?
    - What packages and vulnerabilities does a particular image have?
- Visualize the supply chain and its packages and vulnerabilities of your supply chain.

Components

- Supply Chain Security Tools (SCST) - Scan scans source code and images for their packages and vulnerabilities.
- SCST - Store takes the vulnerability scanning results and stores them.
- Tanzu Insight plug-in provides a CLI to query for packages and vulnerabilities.
- Supply Chain Choreographer in Tanzu Application Platform GUI visualizes the supply chain, including scans, packages, and vulnerabilities.

Next steps

Apply what you have learned:

- Add testing and scanning to your application
- Enable CVE scan results in Supply Chain Choreographer in Tanzu Application Platform GUI

Or learn about:

- Supply chains on Tanzu Application Platform

Or go deeper into scanning on Tanzu Application Platform:

- Scan samples to try the scan and store features as individual one-off scans
- Configure Code Repositories and Image Artifacts to be Scanned
- Code and Image Compliance Policy Enforcement Using Open Policy Agent (OPA)
- How to Create a ScanTemplate
- Viewing and Understanding Scan Status Conditions
- Observing and Troubleshooting
- Tanzu Insight plug-in overview

Troubleshooting

- SCST Scan - Observing and Troubleshooting
- SCST Store - Troubleshooting
- TAP GUI - Troubleshooting

About consuming services on Tanzu Application Platform

This topic describes the key concepts and terms you need to know about consuming services on Tanzu Application Platform (commonly known as TAP).

As part of Tanzu Application Platform, you can work with backing services such as RabbitMQ, PostgreSQL, and MySQL among others. The most common use of services is binding an application workload to a service instance.

Key concepts

When working with services on Tanzu Application Platform, you must be familiar with service instances, service bindings, and resource claims. This section provides a brief overview of each of these key concepts.

Service instances

A **service instance** is a logical grouping of one or more Kubernetes resources that together expose a known capability through a well-defined interface. For example, a theoretical “MySQL” service instance might consist of a `MySQLDatabase` and a `MySQLUser` resource. When considering compatibility of service instances for Tanzu Application Platform, one of the resources of a service instance must adhere to the Service Binding for Kubernetes specification.

Service bindings

Service binding refers to a mechanism in which connectivity information, such as service instance credentials, and connectivity information, such as host and port, are automatically communicated to application workloads. Tanzu Application Platform uses a standard named Service Binding for Kubernetes to implement this mechanism. See this standard to fully understand the services aspect of Tanzu Application Platform.

Resource claims

Resource claims are inspired in part by Persistent Volume Claims. For more information, see the Kubernetes documentation. Resource claims provide a mechanism for users to claim service instances on a cluster, while also decoupling the life cycle of application workloads and service instances.

Services you can use with Tanzu Application Platform

The following list of Kubernetes operators expose APIs that integrate well with Tanzu Application Platform:

1. VMware RabbitMQ for Kubernetes.
2. VMware SQL with Postgres for Kubernetes.
3. VMware SQL with MySQL for Kubernetes.

Compatibility of a service with Tanzu Application Platform ranges on a scale between fully compatible and incompatible. The minimum requirement for compatibility is that there must be a declarative, Kubernetes-based API on which at least one API resource type adheres to the Provisioned Service duck type defined by the Service Binding Specification for Kubernetes in GitHub. This duck type includes any resource type with the following schema:

```
status:
  binding:
    name: # string
```

The value of `.status.binding.name` must point to a `Secret` in the same namespace. The `Secret` contains required credentials and connectivity information for the resource.

Typically, APIs that include these resource types are installed onto the Tanzu Application Platform cluster as Kubernetes operators. These Kubernetes operators provide custom resource definitions (CRDs) and corresponding controllers to reconcile the resources of the CRDs, as is the case with the three Kubernetes operators listed earlier.

For services that do not provide a resource adhering to the Service Binding Specification for Kubernetes, you might still be able to provide configurations allowing such services to integrate with Tanzu Application Platform. For an example of how to do this for Amazon AWS RDS, see the tutorial Integrating cloud services into Tanzu Application Platform.

User roles and responsibilities

It is important to understand the user roles for services on Tanzu Application Platform and the responsibilities assumed by each. The following table describes each user role.

User role	Exists as a default role in Tanzu Application Platform?	Responsibilities
Service operator	Yes - service-operator	• Life cycle management (CRUD) of service instances • Life cycle management (CRUD) of service instance classes • Life cycle management (CRUD) of resource claim policies • View and query for resource claims across namespaces
Application operator	Yes - app-operator	Life cycle management (CRUD) of resource claims
Application developer	Yes - app-editor and app-viewer	Binding service instances to application workloads

Next steps

Apply what you've learned:

- Claim services on Tanzu Application Platform
- Consume services on Tanzu Application Platform

Set up Tanzu Service Mesh

This topic tells you how to set up a Tanzu Application Platform application deployed on Kubernetes with Tanzu Service Mesh (commonly called TSM).

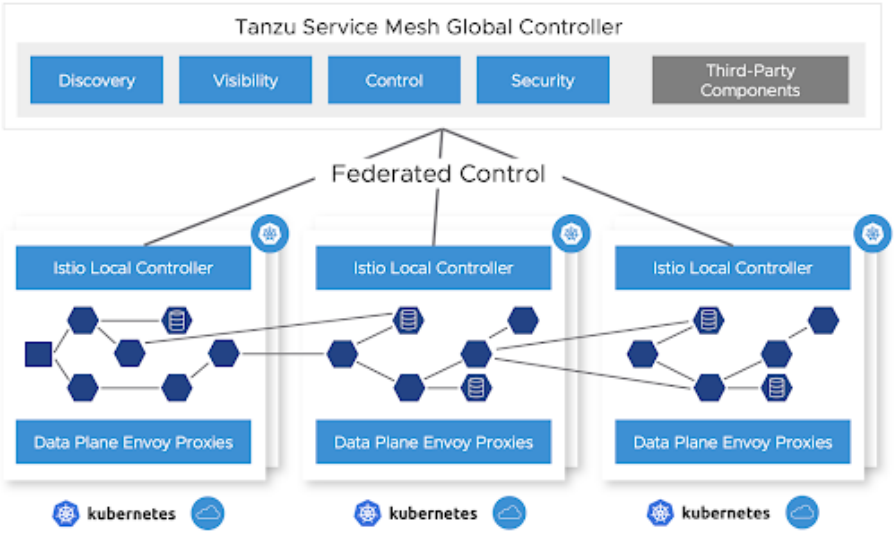
Sample applications are used to demonstrate how a global namespace can provide a network for Kubernetes workloads that are connected and secured within and across clusters, and across clouds.

Prerequisites

Meet the prerequisites, which includes having

- A supported Kubernetes platform
- The correct resource configuration (number of nodes, CPUs, RAM, and so on)
- The required connectivity requirements

Connectivity is only required from your local clusters out to Tanzu Service Mesh and not inwards. This can traverse a corporate proxy as well. In addition, connectivity in the data plane is required between the clusters that must communicate, specifically egress to ingress gateways. No data plane traffic needs to reach the Tanzu Service Mesh software as a service (SaaS) management plane.



Activate your Tanzu Service Mesh subscription

Activate your Tanzu Service Mesh subscription at cloud.vmware.com. After purchasing your Tanzu Service Mesh subscription, the VMware Cloud team sends you instructions. If you don't receive them, you can follow these instructions.

Onboard your clusters to Tanzu Service Mesh as described later in this topic. This deploys the Tanzu Service Mesh local control plane and OSS Istio on your Kubernetes cluster and connects the local control plane to your Tanzu Service Mesh tenant.

Set up Tanzu Application Platform

To enable Tanzu Service Mesh support in Tanzu Application Platform Build clusters:

1. Add the following key to `tap-values.yaml` under the `buildservice` top-level key:

   ```
   buildservice:
    injected_sidecar_support: true
   ```

2. Install Tanzu Application Platform on the run cluster.

End-to-end workload build and deployment scenario

The following sections describe how to build and deploy a workload.

Apply a workload resource to a build cluster

Workloads can be built by using a Tanzu Application Platform supply chain by applying a workload resource to a build cluster. At this time, Tanzu Service Mesh and Tanzu Application Platform cannot use the Knative resources that are the default runtime target when using the `web` resource type.

In Tanzu Application Platform v1.4 and later, two workload types support a Tanzu Service Mesh and Tanzu Application Platform integration: **server** and **worker**.

To work with Tanzu Service Mesh, web workloads must be converted to the `server` or `worker` workload type. Server workloads cause a Kubernetes `Deployment` resource to be created with a `Service` resource that uses port 8080 by default.

1. If the service port that you want is 80 or some other port, add port information to `workload.yaml`. The following example YAML snippets show the changes to make from the `web` to `server` workload type. This is an example before applying the changes:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
    name: hungryman
    labels:
      apps.tanzu.vmware.com/workload-type: web
      app.kubernetes.io/part-of: hungryman-api-gateway
   spec:
    params:
    - name: annotations
   value:
   autoscaling.knative.dev/minScale: "1"
    source:
      git:
        url: https://github.com/gm2552/hungryman.git
      ref:
        branch: main
      subPath: hungryman-api-gateway
   ```

   This is an example modified for Tanzu Service Mesh, which includes the removal of the autoscaling annotation:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
    name: hungryman
    labels:
    apps.tanzu.vmware.com/workload-type: server # modification
    app.kubernetes.io/part-of: hungryman-api-gateway
   spec:
   ```

```
params:
- name: ports # modification
value:
- port: 80 # modification
  containerPort: 8080 # modification
  name: http # modification
source:
git:
    url: https://github.com/gm2552/hungryman.git
  ref:
      branch: main
subPath: hungryman-api-gateway
```

This results in a deployment and a service that listens on port 80 and forwards traffic to port 8080 on the pod's workload container.

2. Submit the modified YAML to your build cluster by running:

```
tanzu apps NAMESPACE apply --file WORKLOAD-YAML-FILE
```

Where:

- `NAMESPACE` is the namespace that the build cluster uses for building.
- `WORKLOAD-YAML-FILE` is the name of your workload YAML file, such as `workload.yaml`.

After your workload is built a `Deliverable` resource is created.

Configure egress for Tanzu Build Service

For Tanzu Build Service to properly work, provide egress to access the registry where Tanzu Build Service writes application images, and define the registry in the `kp_default_repository` key and the Tanzu Application Platform install registry.

Additionally, configure egress for buildpack builds to download any required dependencies. This configuration varies with different buildpacks and language environments. For example, Java builds might need to download dependencies from Maven central.

Create a global namespace

Using the Tanzu Service Mesh portal or API, create a global namespace (GNS) that includes the namespaces where your application components are deployed. For more information, see Global Namespaces

Whether in a single cluster or multiple clusters, or within the same site or across clouds, after you add a namespace selection to the GNS, the services that Tanzu Application Platform deploys are connected based on the GNS configuration for service discovery and connectivity policies.

If a service must be accessible through the ingress from the outside, it can be configured through the public service option in Tanzu Service Mesh or directly through Istio on the clusters where that service resides. It's best practice to configure the service's accessibility through the GNS.

Run cluster deployment

Before deploying a workload to a run cluster, ensure that any prerequisite resources have already been created on the run cluster. This includes concepts such as data, messaging, routing, security services, RBAC, ResourceClaims, and so on.

After a successful build in a build cluster, workloads can be deployed to the run cluster by applying resulting deliverable resources to the run cluster as described in Getting Started with Multicluster Tanzu Application Platform.

Another option is to create a kapp application that references a GitOps repository to include all deliverable resources for a given cluster. See the following example of a kapp definition that points to a GitOps repository:

```
apiVersion: kappctrl.k14s.io/v1alpha1
kind: App
metadata:
  name: deliverable-gitops
  namespace: hungryman
spec:
  serviceAccountName: default
  fetch:
  - git:
    url: https://github.com/gm2552/tap-play-gitops
    ref: origin/deliverables-tap-east01
    subPath: config
  template:
  - ytt: {}
  deploy:
  - kapp: {}
```

The advantage of this model is that applications can be deployed or uninstalled from a cluster by managing the contents of the deliverable resources from within the GitOps repository and enabling a GitOps workflow for application and service change control.

Deployment use case: Hungryman

The following instructions describe an end-to-end process for configuring, building, and deploying the Hungryman application into a Tanzu Service Mesh global namespace.

These instructions use the default configuration of Hungryman, which consists of only needing a single-node RabbitMQ cluster, an in-memory database, and no security. The application is deployed across two Tanzu Application Platform run clusters. It requires the `ytt` command to execute the build and deployment commands.

The configuration resources referenced in this scenario are located in the hungryman-tap-tsm GitHub repository.

Create an initial set of configuration files from the accelerator

This use case deployment includes a pre-built set of configuration files in a Git repository. However, they were created from a set of configuration files by using a bootstrapped process that uses the Hungryman accelerator, and were later modified.

For reference, you can create an initial set of configuration files from the Hungryman accelerator, which is available in Tanzu Application Platform v1.3.

This section does not include instructions for modifying the configuration files from the accelerator into configuration files used in a later section.

From the accelerator, accept all of the default options with the following exceptions:

- **Workload namespace:** Update this field with the name of the namespace you will use to build the application in your build cluster
- **Service namespace:** Update this field with the name of the namespace you will use to deploy a RabbitMQ cluster on your Tanzu Application Platform run cluster

Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in this command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
```

Where `WORKLOAD-NAMESPACE` is the name of your build namespace

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/workloads.yaml
\
-v workloadNamespace=workloads | kubectl apply -f-
```

If you are using a GitOps workflow with your build cluster, after the workloads are built the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the config-writer pods that commit deployment information to the GtiOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in the instructions.

Install service claim resources on the cluster

Hungryman requires a RabbitMQ cluster installed on your run cluster. You must install RabbitMQ on the same run cluster that is named `RunCluster01` in the following deployment section. Additionally, you must install service claim resources on this cluster.

1. If you haven't already done so, install the RabbitMQ Cluster Operator on the run cluster by running:

   ```
   kubectl apply -f "https://github.com/rabbitmq/cluster-operator/releases/downloa
   d/v1.13.1/cluster-operator.yml"
   ```

2. Spin up an instance of a RabbitMQ cluster by running:

   ```
   kubectl create ns SERVICE-NAMESPACE

   ytt -f rmqCluster.yaml -v serviceNamespace=SERVICE-NAMESPACE | kubectl apply -f
   -
   ```

   Where `SERVICE-NAMESPACE` is the namespace of where you want to deploy your RabbitMQ cluster

   For example:

   ```
   kubectl create ns service-instances

   ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/rmqClust
   er.yaml -v \
   serviceNamespace=service-instances | kubectl apply -f-
   ```

3. Create service toolkit resources for the RabbitMQ class and resource claim by running:

   ```
   ytt -f rmqResourceClaim.yaml -v serviceNamespace=SERVICE-NAMESPACE -v \
   workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
   ```

   Where `SERVICE-NAMESPACE` and `WORKLOAD-NAMESPACE` are the namespaces where you deployed your RabbitMQ cluster and the namespace where the application service will run.

   For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/rmqResou
rceClaim.yaml \
-v serviceNamespace=service-instances -v workloadNamespace=hungryman | kubectl
apply -f-
```

Run cluster deployment

Workloads are deployed to the run cluster using deliverable resources. This section applies the deliverable resources directly to the run clusters instead of using a kapp application.

This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS Hungryman. These example clusters are named `RunCluster01` and `RunCluster02`. The majority of the workload is deployed to `RunCluster01` while the crawler workload is deployed to `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and needs to reference this repository in the following commands.

Deploy the workloads to the run clusters by running these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

To run this deployment on cluster `RunCluster01`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/cluster01Delive
rables.yaml -v \
workloadNamespace=hungryman -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

To run this deployment on cluster `RunCluster02`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/cluster02Delive
rables.yaml -v \
workloadNamespace=hungryman -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

You can create an Istio ingress resource on `RunCluster01` if you do not plan on using the GNS capabilities to expose the application to external networks.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS

configuration is out of the scope of this topic.

Create the ingress by running:

```
ytt -f ingress.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v domainName=DOMAIN-NAME
| kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed
- `DOMAIN-NAME` is the public domain that will host your application

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/ingress.yaml -v
\
workloadNamespace=hungryman -v domainName=tsmdemo.perfect300rock.com | kubectl apply -
f-
```

Create a global namespace

The example clusters have the names `RunCluster01` and `RunCluster02`, and they assume the workload and service namespaces of Hungryman and service-instances, respectively.

1. Open the Tanzu Service Mesh console and create a new GNS.
2. Configure the following settings in each step:
    1. General details
        - **GNS Name:** hungryman
        - **Domain:** hungryman.lab
    2. Namespace mapping
        - Namespace mapping Rule 1
            - **Cluster name:** RunCluster01
            - **Namespace:** hungryman
        - Namespace Mapping Rule 2
            - **Cluster name:** RunCluster02
            - **Namespace:** hungryman
        - Namespace Mapping Rule 3
            - **Cluster name:** RunCluster01
            - **Namespace:** service-instances
    3. Autodiscovery. Use the default settings.
    4. Public services
        - **Service name:** hungryman
        - **Service port:** 80
        - **Public URL:** http hungryman . Select a domain.
    5. Global server load balancing and resiliency. Use the default settings.

You can now access the Hungryman application with the URL configured earlier.

Deployment use case: ACME Fitness Store

The following instructions describe an end-to-end process for configuring, building, and deploying the ACME Fitness Store application into a Tanzu Service Mesh GNS. In this use case, the application is deployed across two Tanzu Application Platform run clusters. ytt is used to run the build and deployment commands.

The configuration resources referenced in this scenario are in the acme-fitness-tap-tsm Git repository.

Deploy AppSSO

ACME requires the use of an AppSSO authorization server and client registration resource. Install these resources on the same run cluster that is named `RunCluster01` in the deployment section.

1. Deploy the authorization server instance by running:

   ```
   ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE \
   -v devDefaultAccountUsername=DEV-DEFAULT-ACCOUNT-USERNAME -v \
   devDefaultAccountPassword=DEV-DEFAULT-ACCOUNT-PASSWORD | kubectl apply -f-
   ```

   Where:

   - `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
   - `DEV-DEFAULT-ACCOUNT-USERNAME` is the user name for the ACME application authentication
   - `DEV-DEFAULT-ACCOUNT-PASSWORD` is the password for the ACME application authentication

   For example:

   ```
   ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/appSS
   OInstance.yaml -v \
   workloadNamespace=acme -v devDefaultAccountUsername=acme -v \
   devDefaultAccountPassword=fitness | kubectl apply -f-
   ```

2. Create a `ClientRegistration` resource by running:

   ```
   ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
   appSSORedirectURI=APP-SSO-REDIRECT-URI | kubectl apply -f-
   ```

   Where:

   - `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed.
   - `APP-SSO-REDIRECT-URI` is the public URI that the authorization server redirects to after a login

   For example:

   ```
   ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/clien
   tRegistrationResourceClaim.yaml \
   -v workloadNamespace=acme -v \
   appSSORedirectURI=http://acme-fitness.tsmdemo.perfect300rock.com/login/oauth2/c
   ode/sso | kubectl apply -f-
   ```

3. Obtain the appSSO Issuer URI by running:

   ```
   kubectl get authserver -n WORKLOAD-NAMESPACE
   ```

   Where `WORKLOAD-NAMESPACE` is the name of the namespace where the workloads will be deployed.

4. Record the Issuer URI because you need it for the next section.

Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in the following command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appSSOIssuerURI=APP-SSO-ISSUER-URL | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-SSO-ISSUER-URL` is the URL of the AppSSO authorization server that you deployed earlier

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/workloads.ya
ml -v \
workloadNamespace=workloads -v \
appSSOIssuerURI=http://appsso-acme-fitness.acme.tsmdemo.perfect300rock.com | kubectl a
pply -f-
```

If you are using a GitOps workflow with your build cluster then, after building the workloads, the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the `config-writer` pods that commit deployment information to the GitOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in these instructions.

Create the Istio ingress resources

The authorization server requires a publicly accessible URL and must be available before the Spring Cloud Gateway can deploy properly. The authorization server is deployed at the URI authserver app domain.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the Istio ingress resources by running:

```
ytt -f istioGateway.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appDomainName=APP-DOMAIN | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-DOMAIN` is the application's DNS domain

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/istioGatewa
y.yaml -v \
workloadNamespace=acme -v appDomainName=tsmdemo.perfect300rock.com | kubectl apply -f-
```

Deploy Redis

A Redis instance is needed for caching the ACME fitness store cart service. Deploy the Redis instance by running:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -
v \
workloadNamespace=WORKLOAD-NAMESPACE -v redisPassword=REDIS-PASSWORD | kb apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `REDIS-PASSWORD` is your password

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -
v \
workloadNamespace=acme -v redisPassword=fitness | kubectl apply -f-
```

Run cluster deployment

Workloads are deployed to the run cluster by using deliverable resources. In this section you apply the deliverable resources directly to the run clusters, instead of using a kapp application. This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS ACME. In this example these clusters are named `RunCluster01` and `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and need to reference this repository in the following commands.

To deploy the workloads to the run clusters, run these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

For the `RunCluster01` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster01Del
iverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

For the `RunCluster02` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster02Del
iverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

Deploy Spring Cloud Gateway

The following sections describe how to deploy Spring Cloud Gateway.

Install the Spring Cloud Gateway package

The section requires the Spring Cloud Gateway for Kubernetes package to be installed on `RunCluster01`. If Spring Cloud Gateway is already installed on the run cluster, skip these install steps.

In Tanzu Application Platform v1.5 and later, Spring Cloud Gateway is included as an optional package in the Tanzu Application Platform Carvel bundle. Install the Spring Cloud Gateway package with the default settings by using this Tanzu CLI template:

```
tanzu package install scg --package spring-cloud-gateway.tanzu.vmware.com \
-version VERSION-NUMBER -n TAP-INSTALL-NAMESPACE
```

For example:

```
tanzu package install scg --package spring-cloud-gateway.tanzu.vmware.com \
--version 2.0.0-tap.3 -n tap-install
```

Configure the Spring Cloud Gateway instance and route

The Tanzu Application Platform fork of the ACME fitness store uses Spring Cloud Gateway for routing API classes from the web front end to the microservices.

Caution

The Spring Cloud Gateway `spec.service.name` configuration was not built with multicluster or cross-cluster support. The configuration for the gateway routes currently implements a workaround, which is brittle in terms of where certain services are deployed. Future releases of the gateway might have better support for this use case.

Deploy the gateway and applicable routes by running:

```
ytt -f scgInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

```
ytt -f scgRoutes.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgInstance.
yaml -v \
workloadNamespace=acme | kubectl apply -f-
```

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgRoutes.ya
ml -v \
workloadNamespace=acme | kubectl apply -f-
```

Create a global namespace

The example clusters are named `RunCluster01` and `RunCluster02`, and they assume a workload namespace of ACME.

1. Open the Tanzu Service Mesh console and create a new global namespace.
2. Configure the following settings in each step:
    1. General details
        - **GNS name:** acme-tap
        - **Domain:** acme-tap.lab
    2. Namespace mapping
        - Namespace mapping Rule 1
            - **Cluster name:** RunCluster01
            - **Namespace:** acme
        - Namespace Mapping Rule 2
            - **Cluster name:** RunCluster02
            - **Namespace:** acme
    3. Autodiscovery. Use the default settings.
    4. Public Services
        - No Public service
    5. Global server load-balancing and resiliency. Use the default settings.

You can access the application by going to the URL `http://acme-fitness`.

Set up Tanzu Service Mesh

This topic tells you how to set up a Tanzu Application Platform application deployed on Kubernetes with Tanzu Service Mesh (commonly called TSM).

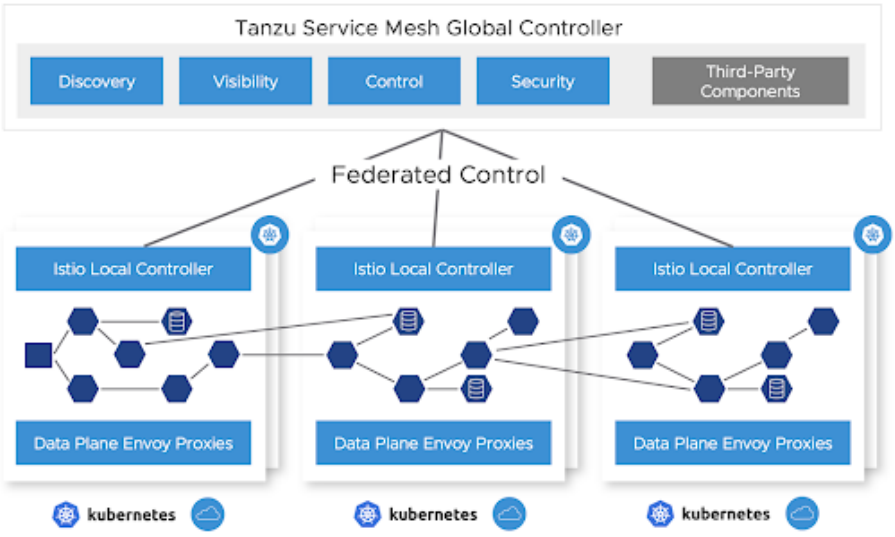
Sample applications are used to demonstrate how a global namespace can provide a network for Kubernetes workloads that are connected and secured within and across clusters, and across clouds.

Prerequisites

Meet the prerequisites, which includes having

- A supported Kubernetes platform
- The correct resource configuration (number of nodes, CPUs, RAM, and so on)
- The required connectivity requirements

Connectivity is only required from your local clusters out to Tanzu Service Mesh and not inwards. This can traverse a corporate proxy as well. In addition, connectivity in the data plane is required between the clusters that must communicate, specifically egress to ingress gateways. No data plane traffic needs to reach the Tanzu Service Mesh software as a service (SaaS) management plane.



Activate your Tanzu Service Mesh subscription

Activate your Tanzu Service Mesh subscription at cloud.vmware.com. After purchasing your Tanzu Service Mesh subscription, the VMware Cloud team sends you instructions. If you don't receive them, you can follow these instructions.

Onboard your clusters to Tanzu Service Mesh as described later in this topic. This deploys the Tanzu Service Mesh local control plane and OSS Istio on your Kubernetes cluster and connects the local control plane to your Tanzu Service Mesh tenant.

Set up Tanzu Application Platform

To enable Tanzu Service Mesh support in Tanzu Application Platform Build clusters:

1. Add the following key to `tap-values.yaml` under the `buildservice` top-level key:

   ```
   buildservice:
    injected_sidecar_support: true
   ```

2. Install Tanzu Application Platform on the run cluster.

End-to-end workload build and deployment scenario

The following sections describe how to build and deploy a workload.

Apply a workload resource to a build cluster

Workloads can be built by using a Tanzu Application Platform supply chain by applying a workload resource to a build cluster. At this time, Tanzu Service Mesh and Tanzu Application Platform cannot use the Knative resources that are the default runtime target when using the `web` resource type.

In Tanzu Application Platform v1.4 and later, two workload types support a Tanzu Service Mesh and Tanzu Application Platform integration: **server** and **worker**.

To work with Tanzu Service Mesh, web workloads must be converted to the `server` or `worker` workload type. Server workloads cause a Kubernetes `Deployment` resource to be created with a `Service` resource that uses port 8080 by default.

1. If the service port that you want is 80 or some other port, add port information to `workload.yaml`. The following example YAML snippets show the changes to make from the

`web` to `server` workload type. This is an example before applying the changes:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
 name: hungryman
 labels:
   apps.tanzu.vmware.com/workload-type: web
   app.kubernetes.io/part-of: hungryman-api-gateway
spec:
 params:
 - name: annotations
value:
autoscaling.knative.dev/minScale: "1"
 source:
   git:
     url: https://github.com/gm2552/hungryman.git
   ref:
     branch: main
   subPath: hungryman-api-gateway
```

This is an example modified for Tanzu Service Mesh, which includes the removal of the autoscaling annotation:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
 name: hungryman
 labels:
 apps.tanzu.vmware.com/workload-type: server # modification
 app.kubernetes.io/part-of: hungryman-api-gateway
spec:
 params:
 - name: ports # modification
 value:
 - port: 80 # modification
   containerPort: 8080 # modification
   name: http # modification
 source:
 git:
     url: https://github.com/gm2552/hungryman.git
   ref:
       branch: main
 subPath: hungryman-api-gateway
```

This results in a deployment and a service that listens on port 80 and forwards traffic to port 8080 on the pod's workload container.

2. Submit the modified YAML to your build cluster by running:

```
tanzu apps NAMESPACE apply --file WORKLOAD-YAML-FILE
```

Where:

- `NAMESPACE` is the namespace that the build cluster uses for building.
- `WORKLOAD-YAML-FILE` is the name of your workload YAML file, such as `workload.yaml`.

After your workload is built a `Deliverable` resource is created.

Configure egress for Tanzu Build Service

For Tanzu Build Service to properly work, provide egress to access the registry where Tanzu Build Service writes application images, and define the registry in the `kp_default_repository` key and the Tanzu Application Platform install registry.

Additionally, configure egress for buildpack builds to download any required dependencies. This configuration varies with different buildpacks and language environments. For example, Java builds might need to download dependencies from Maven central.

Create a global namespace

Using the Tanzu Service Mesh portal or API, create a global namespace (GNS) that includes the namespaces where your application components are deployed. For more information, see Global Namespaces

Whether in a single cluster or multiple clusters, or within the same site or across clouds, after you add a namespace selection to the GNS, the services that Tanzu Application Platform deploys are connected based on the GNS configuration for service discovery and connectivity policies.

If a service must be accessible through the ingress from the outside, it can be configured through the public service option in Tanzu Service Mesh or directly through Istio on the clusters where that service resides. It's best practice to configure the service's accessibility through the GNS.

Run cluster deployment

Before deploying a workload to a run cluster, ensure that any prerequisite resources have already been created on the run cluster. This includes concepts such as data, messaging, routing, security services, RBAC, ResourceClaims, and so on.

After a successful build in a build cluster, workloads can be deployed to the run cluster by applying resulting deliverable resources to the run cluster as described in Getting Started with Multicluster Tanzu Application Platform.

Another option is to create a kapp application that references a GitOps repository to include all deliverable resources for a given cluster. See the following example of a kapp definition that points to a GitOps repository:

```
apiVersion: kappctrl.k14s.io/v1alpha1
kind: App
metadata:
  name: deliverable-gitops
  namespace: hungryman
spec:
  serviceAccountName: default
  fetch:
  - git:
    url: https://github.com/gm2552/tap-play-gitops
    ref: origin/deliverables-tap-east01
    subPath: config
  template:
  - ytt: {}
  deploy:
  - kapp: {}
```

The advantage of this model is that applications can be deployed or uninstalled from a cluster by managing the contents of the deliverable resources from within the GitOps repository and enabling a GitOps workflow for application and service change control.

Deployment use case: Hungryman

The following instructions describe an end-to-end process for configuring, building, and deploying the Hungryman application into a Tanzu Service Mesh global namespace.

These instructions use the default configuration of Hungryman, which consists of only needing a single-node RabbitMQ cluster, an in-memory database, and no security. The application is deployed across two Tanzu Application Platform run clusters. It requires the `ytt` command to execute the build and deployment commands.

The configuration resources referenced in this scenario are located in the hungryman-tap-tsm GitHub repository.

Create an initial set of configuration files from the accelerator

This use case deployment includes a pre-built set of configuration files in a Git repository. However, they were created from a set of configuration files by using a bootstrapped process that uses the Hungryman accelerator, and were later modified.

For reference, you can create an initial set of configuration files from the Hungryman accelerator, which is available in Tanzu Application Platform v1.3.

This section does not include instructions for modifying the configuration files from the accelerator into configuration files used in a later section.

From the accelerator, accept all of the default options with the following exceptions:

- **Workload namespace:** Update this field with the name of the namespace you will use to build the application in your build cluster
- **Service namespace:** Update this field with the name of the namespace you will use to deploy a RabbitMQ cluster on your Tanzu Application Platform run cluster

Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in this command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
```

Where `WORKLOAD-NAMESPACE` is the name of your build namespace

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/workloads.yaml
\
-v workloadNamespace=workloads | kubectl apply -f-
```

If you are using a GitOps workflow with your build cluster, after the workloads are built the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the config-writer pods that commit deployment information to the GtiOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in the instructions.

Install service claim resources on the cluster

Hungryman requires a RabbitMQ cluster installed on your run cluster. You must install RabbitMQ on the same run cluster that is named `RunCluster01` in the following deployment section. Additionally, you must install service claim resources on this cluster.

1. If you haven't already done so, install the RabbitMQ Cluster Operator on the run cluster by running:

   ```
   kubectl apply -f "https://github.com/rabbitmq/cluster-operator/releases/downloa
   d/v1.13.1/cluster-operator.yml"
   ```

2. Spin up an instance of a RabbitMQ cluster by running:

   ```
   kubectl create ns SERVICE-NAMESPACE

   ytt -f rmqCluster.yaml -v serviceNamespace=SERVICE-NAMESPACE | kubectl apply -f
   -
   ```

   Where `SERVICE-NAMESPACE` is the namespace of where you want to deploy your RabbitMQ cluster

   For example:

   ```
   kubectl create ns service-instances

   ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/rmqClust
   er.yaml -v \
   serviceNamespace=service-instances | kubectl apply -f-
   ```

3. Create service toolkit resources for the RabbitMQ class and resource claim by running:

   ```
   ytt -f rmqResourceClaim.yaml -v serviceNamespace=SERVICE-NAMESPACE -v \
   workloadNamespace=WORKLOAD-NAMESPACE | kubectl apply -f-
   ```

   Where `SERVICE-NAMESPACE` and `WORKLOAD-NAMESPACE` are the namespaces where you deployed your RabbitMQ cluster and the namespace where the application service will run.

   For example:

   ```
   ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/rmqResou
   rceClaim.yaml \
   -v serviceNamespace=service-instances -v workloadNamespace=hungryman | kubectl
   apply -f-
   ```

Run cluster deployment

Workloads are deployed to the run cluster using deliverable resources. This section applies the deliverable resources directly to the run clusters instead of using a kapp application.

This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS Hungryman. These example clusters are named `RunCluster01` and `RunCluster02`. The majority of the workload is deployed to `RunCluster01` while the crawler workload is deployed to `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and needs to reference this repository in the following commands.

Deploy the workloads to the run clusters by running these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository

- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads are deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

To run this deployment on cluster `RunCluster01`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/cluster01Delive
rables.yaml -v \
workloadNamespace=hungryman -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

To run this deployment on cluster `RunCluster02`, for example, you run:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/cluster02Delive
rables.yaml -v \
workloadNamespace=hungryman -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

You can create an Istio ingress resource on `RunCluster01` if you do not plan on using the GNS capabilities to expose the application to external networks.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the ingress by running:

```
ytt -f ingress.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v domainName=DOMAIN-NAME
| kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed
- `DOMAIN-NAME` is the public domain that will host your application

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/hungryman-tap-tsm/main/ingress.yaml -v
\
workloadNamespace=hungryman -v domainName=tsmdemo.perfect300rock.com | kubectl apply -
f-
```

Create a global namespace

The example clusters have the names `RunCluster01` and `RunCluster02`, and they assume the workload and service namespaces of Hungryman and service-instances, respectively.

1. Open the Tanzu Service Mesh console and create a new GNS.
2. Configure the following settings in each step:
    1. General details

        - **GNS Name:** hungryman
        - **Domain:** hungryman.lab
    2. Namespace mapping
        - Namespace mapping Rule 1
            - **Cluster name:** RunCluster01
            - **Namespace:** hungryman
        - Namespace Mapping Rule 2
            - **Cluster name:** RunCluster02
            - **Namespace:** hungryman
        - Namespace Mapping Rule 3
            - **Cluster name:** RunCluster01
            - **Namespace:** service-instances
    3. Autodiscovery. Use the default settings.
    4. Public services
        - **Service name:** hungryman
        - **Service port:** 80
        - **Public URL:** http hungryman . Select a domain.
    5. Global server load balancing and resiliency. Use the default settings.

You can now access the Hungryman application with the URL configured earlier.

Deployment use case: ACME Fitness Store

The following instructions describe an end-to-end process for configuring, building, and deploying the ACME Fitness Store application into a Tanzu Service Mesh GNS. In this use case, the application is deployed across two Tanzu Application Platform run clusters. ytt is used to run the build and deployment commands.

The configuration resources referenced in this scenario are in the acme-fitness-tap-tsm Git repository.

Deploy AppSSO

ACME requires the use of an AppSSO authorization server and client registration resource. Install these resources on the same run cluster that is named `RunCluster01` in the deployment section.

1. Deploy the authorization server instance by running:

    ```
    ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE \
    -v devDefaultAccountUsername=DEV-DEFAULT-ACCOUNT-USERNAME -v \
    devDefaultAccountPassword=DEV-DEFAULT-ACCOUNT-PASSWORD | kubectl apply -f-
    ```

    Where:

    - `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
    - `DEV-DEFAULT-ACCOUNT-USERNAME` is the user name for the ACME application authentication
    - `DEV-DEFAULT-ACCOUNT-PASSWORD` is the password for the ACME application authentication

    For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/appSS
OInstance.yaml -v \
workloadNamespace=acme -v devDefaultAccountUsername=acme -v \
devDefaultAccountPassword=fitness | kubectl apply -f-
```

2. Create a `ClientRegistration` resource by running:

   ```
   ytt -f appSSOInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
   appSSORedirectURI=APP-SSO-REDIRECT-URI | kubectl apply -f-
   ```

   Where:

   - `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed.
   - `APP-SSO-REDIRECT-URI` is the public URI that the authorization server redirects to after a login

   For example:

   ```
   ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/clien
   tRegistrationResourceClaim.yaml \
   -v workloadNamespace=acme -v \
   appSSORedirectURI=http://acme-fitness.tsmdemo.perfect300rock.com/login/oauth2/c
   ode/sso | kubectl apply -f-
   ```

3. Obtain the appSSO Issuer URI by running:

   ```
   kubectl get authserver -n WORKLOAD-NAMESPACE
   ```

   Where `WORKLOAD-NAMESPACE` is the name of the namespace where the workloads will be deployed.

4. Record the Issuer URI because you need it for the next section.

Apply the workload resources to your build cluster

To build the application services, run the following command to apply the workload resources to your build cluster. You can also clone or fork the repository in the following command to either use the YAML files locally or point to your own Git repository.

```
ytt -f workloads.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appSSOIssuerURI=APP-SSO-ISSUER-URL | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-SSO-ISSUER-URL` is the URL of the AppSSO authorization server that you deployed earlier

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/workloads.ya
ml -v \
workloadNamespace=workloads -v \
appSSOIssuerURI=http://appsso-acme-fitness.acme.tsmdemo.perfect300rock.com | kubectl a
pply -f-
```

If you are using a GitOps workflow with your build cluster then, after building the workloads, the deployment information is pushed to your GitOps repository.

If you follow these instructions without pull requests in the GitOps workflow, the `config-writer` pods that commit deployment information to the GitOps repository might fail because of concurrency conflicts. A workaround for this is to delete the failed workloads from the build cluster and re-run the command provided in these instructions.

Create the Istio ingress resources

The authorization server requires a publicly accessible URL and must be available before the Spring Cloud Gateway can deploy properly. The authorization server is deployed at the URI authserver app domain.

You must create a domain name system address (DNS A) record in your DNS provider's configuration tool to point to the Istio load-balanced IP address of `RunCluster01`. The DNS configuration is out of the scope of this topic.

Create the Istio ingress resources by running:

```
ytt -f istioGateway.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
appDomainName=APP-DOMAIN | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the name of your build namespace
- `APP-DOMAIN` is the application's DNS domain

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/istioGatewa
y.yaml -v \
workloadNamespace=acme -v appDomainName=tsmdemo.perfect300rock.com | kubectl apply -f-
```

Deploy Redis

A Redis instance is needed for caching the ACME fitness store cart service. Deploy the Redis instance by running:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -
v \
workloadNamespace=WORKLOAD-NAMESPACE -v redisPassword=REDIS-PASSWORD | kb apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `REDIS-PASSWORD` is your password

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/redis.yaml -
v \
workloadNamespace=acme -v redisPassword=fitness | kubectl apply -f-
```

Run cluster deployment

Workloads are deployed to the run cluster by using deliverable resources. In this section you apply the deliverable resources directly to the run clusters, instead of using a kapp application. This deployment assumes that two clusters are part of the Tanzu Service Mesh GNS ACME. In this example these clusters are named `RunCluster01` and `RunCluster02`.

The deliverable objects reference the GitOps repository, where the build cluster has written deployment information, and need to reference this repository in the following commands.

To deploy the workloads to the run clusters, run these commands against their respective clusters:

```
ytt -f cluster01Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

```
ytt -f cluster02Deliverables.yaml -v workloadNamespace=WORKLOAD-NAMESPACE -v \
gitOpsSecret=GIT-OPS-SECRET -v gitOpsRepo=GIT-OPS-REPO | kubectl apply -f-
```

Where:

- `WORKLOAD-NAMESPACE` is the namespace where the workloads will be deployed
- `GIT-OPS-SECRET` is the GitOps secret used to access the GitOps repository
- `GIT-OPS-REPO` is the URL of the GitOps repository where the build cluster wrote out deployment configuration information

For the `RunCluster01` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster01Del
iverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

For the `RunCluster02` example, run:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/cluster02Del
iverables.yaml \
-v workloadNamespace=acme -v gitOpsSecret=tap-play-gitops-secret -v \
gitOpsRepo=https://github.com/gm2552/tap-play-gitops.git | kubectl apply -f-
```

Deploy Spring Cloud Gateway

The following sections describe how to deploy Spring Cloud Gateway.

Install the Spring Cloud Gateway package

The section requires the Spring Cloud Gateway for Kubernetes package to be installed on `RunCluster01`. If Spring Cloud Gateway is already installed on the run cluster, skip these install steps.

In Tanzu Application Platform v1.5 and later, Spring Cloud Gateway is included as an optional package in the Tanzu Application Platform Carvel bundle. Install the Spring Cloud Gateway package with the default settings by using this Tanzu CLI template:

```
tanzu package install scg --package spring-cloud-gateway.tanzu.vmware.com \
-version VERSION-NUMBER -n TAP-INSTALL-NAMESPACE
```

For example:

```
tanzu package install scg --package spring-cloud-gateway.tanzu.vmware.com \
--version 2.0.0-tap.3 -n tap-install
```

Configure the Spring Cloud Gateway instance and route

The Tanzu Application Platform fork of the ACME fitness store uses Spring Cloud Gateway for routing API classes from the web front end to the microservices.

Caution

The Spring Cloud Gateway `spec.service.name` configuration was not built with multicluster or cross-cluster support. The configuration for the gateway routes currently implements a workaround, which is brittle in terms of where certain services are deployed. Future releases of the gateway might have better support for this use case.

Deploy the gateway and applicable routes by running:

```
ytt -f scgInstance.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

```
ytt -f scgRoutes.yaml -v workloadNamespace=WORKLOAD-NAMESPACE
```

Where `WORKLOAD-NAMESPACE` is the namespace where the workload is deployed.

For example:

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgInstance.
yaml -v \
workloadNamespace=acme | kubectl apply -f-
```

```
ytt -f https://raw.githubusercontent.com/gm2552/acme-fitness-tap-tsm/main/scgRoutes.ya
ml -v \
workloadNamespace=acme | kubectl apply -f-
```

Create a global namespace

The example clusters are named `RunCluster01` and `RunCluster02`, and they assume a workload namespace of ACME.

1. Open the Tanzu Service Mesh console and create a new global namespace.
2. Configure the following settings in each step:
    1. General details
        - **GNS name:** acme-tap
        - **Domain:** acme-tap.lab
    2. Namespace mapping
        - Namespace mapping Rule 1
            - **Cluster name:** RunCluster01
            - **Namespace:** acme
        - Namespace Mapping Rule 2
            - **Cluster name:** RunCluster02

        - **Namespace:** acme
    3. Autodiscovery. Use the default settings.
    4. Public Services
        - No Public service
    5. Global server load-balancing and resiliency. Use the default settings.

You can access the application by going to the URL `http://acme-fitness`.

Overview of workloads

This topic provides you with an overview of workload types in Tanzu Application Platform (commonly known as TAP).

Workload features

Tanzu Application Platform allows you to quickly build and test applications regardless of your familiarity with Kubernetes.

You can turn source code into a workload that runs in a container with a URL. You can also use supply chains to build applications that process work from a message queue, or provide arbitrary network services.

A workload allows you to choose application specifications, such as repository location, environment variables, service binding, and so on. For more information about workload creation and management, see Commands Details.

The Out of the Box supply chains support a range of workload types, including

- Scalable web applications (`web`)
- Traditional application servers (`server`)
- Background applications (`worker`)
- Serverless functions

You can use a collection of workloads of different types to deploy microservices that function as a logical application. Alternatively, you can deploy your entire application as a single monolith.

If you build your own supply chains, you can define additional deployment methods beyond those in the Out of the Box supply-chain templates.

Available workload types

When using the Out of the Box supply chain, the `apps.tanzu.vmware.com/workload-type` annotation selects which style of deployment is suitable for your application. The valid values are:

Type	Description	Indicators
`web`	Scalable web applications	• Scales based on request load • Automatically exposed by HTTP Ingress • Does not perform background work • Works with Service Bindings • Stateless • Quick startup time

Type	Description	Indicators
`server`	Traditional applications	• Provides HTTP or TCP services on the network • Exposed by external Ingress or LoadBalancer settings • Might perform background work from a queue • Works with Service Bindings • Fixed scaling, no disk persistence • Startup time not an issue
`worker`	Background applications	• Does not provide network services • Not exposed externally as a network service • Performs background work from a queue • Works with Service Bindings • Fixed scaling, no disk persistence • Startup time not an issue

Overview of workloads

This topic provides you with an overview of workload types in Tanzu Application Platform (commonly known as TAP).

Workload features

Tanzu Application Platform allows you to quickly build and test applications regardless of your familiarity with Kubernetes.

You can turn source code into a workload that runs in a container with a URL. You can also use supply chains to build applications that process work from a message queue, or provide arbitrary network services.

A workload allows you to choose application specifications, such as repository location, environment variables, service binding, and so on. For more information about workload creation and management, see Commands Details.

The Out of the Box supply chains support a range of workload types, including

- Scalable web applications (`web`)
- Traditional application servers (`server`)
- Background applications (`worker`)
- Serverless functions

You can use a collection of workloads of different types to deploy microservices that function as a logical application. Alternatively, you can deploy your entire application as a single monolith.

If you build your own supply chains, you can define additional deployment methods beyond those in the Out of the Box supply-chain templates.

Available workload types

When using the Out of the Box supply chain, the `apps.tanzu.vmware.com/workload-type` annotation selects which style of deployment is suitable for your application. The valid values are:

Type	Description	Indicators
`web`	Scalable web applications	• Scales based on request load • Automatically exposed by HTTP Ingress • Does not perform background work • Works with Service Bindings • Stateless • Quick startup time
`server`	Traditional applications	• Provides HTTP or TCP services on the network • Exposed by external Ingress or LoadBalancer settings • Might perform background work from a queue • Works with Service Bindings • Fixed scaling, no disk persistence • Startup time not an issue
`worker`	Background applications	• Does not provide network services • Not exposed externally as a network service • Performs background work from a queue • Works with Service Bindings • Fixed scaling, no disk persistence • Startup time not an issue

Use web workloads

This topic tells you how to use the `web` workload type in Tanzu Application Platform (commonly known as TAP).

Overview

The `web` workload type allows you to deploy web applications on Tanzu Application Platform. Using an application workload specification, you can turn source code into a scalable, stateless application that runs in a container with an automatically-assigned URL. This type of application is often called serverless, and is deployed using Knative.

The `web` workload type is suitable for modern stateless web applications that follow the twelve-factor app methodology and have the following characteristics:

- Perform all work through HTTP requests, including gRPC and WebSocket
- Do not perform work except when processing a request
- Start up quickly
- Store state in external databases instead of storing state locally

Applications using the `web` workload type have the following features:

- Automatic request-based scaling, including scale-to-zero
- Automatic URL provisioning and optional certificate provisioning
- Automatic health-check definitions, if not provided by a convention
- Blue-green application rollouts

When creating a workload with the `tanzu apps workload create` command, you can use the `--type=web` argument to select the `web` workload type. For more information, see Use the web Workload Type later in this topic.

You can also use the `apps.tanzu.vmware.com/workload-type:web` label in the YAML workload description to support this deployment type.

Use the `web` workload type

The `tanzu-java-web-app` workload mentioned in Deploy an app on Tanzu Application Platform is a good match for the `web` workload type. It is a good match because it serves HTTP requests and does not perform any background processing.

You can experiment with the differences between the `web` and `server` workload types by changing the workload type. To change the workload type run:

```
tanzu apps workload apply tanzu-java-web-app --type=server
```

After changing the workload type to `server`, the application does not auto-scale or expose an external URL. For more information about the server workload type, see Use Server workloads.

Switch back to the `web` workload by running:

```
tanzu apps workload apply tanzu-java-web-app --type=web
```

Use this to test which applications can function well as serverless web applications, and which are more suited to the `server` application style.

Calling `web` workloads within a cluster

When a `web` workload type is created, a Knative service is deployed to the cluster. To access your application, you need the URL for the route created by the Knative Service. Obtain it by running one of these commands:

```
tanzu apps workload get WORKLOAD-NAME --namespace DEVELOPER-NAMESPACE
kubectl get ksvc WORKLOAD-NAME -n YOUR-DEVELOPER-NAMESPACE -ojsonpath="{status.addres
s.url}"
```

When calling a Knative service, both the Service name and namespace are required. This behavior is distinct from `server` type workloads, which do not rely on the namespace name to establish service to service communication between applications within the same namespace.

Example of service to service communication for `web` and `server` workloads

You have three applications deployed to the namespace called `dev-namespace`:

1. A `server` type workload named `server-workload`
2. A `web` type workload named `web-workload`
3. A pod running the busybox image with curl, named `busybox`

Open a shell to the running container of the `busybox` pod and send requests to the `server` and `web` workloads using curl. Specify the namespace for both, as follows:

```
kubectl exec busybox -n dev-namespace -- curl server-workload.dev-namespace.svc.cluste
r.local -v
```

```
kubectl exec busybox -n dev-namespace -- curl web-workload.dev-namespace.svc.cluster.l
ocal -v
```

Use server workloads

This topic tells you how to use the `server` workload type in Tanzu Application Platform (commonly known as TAP).

Overview

The `server` workload type allows you to deploy traditional network applications on Tanzu Application Platform.

Using an application workload specification, you can build and deploy application source code to a manually-scaled Kubernetes deployment which exposes an in-cluster Service endpoint. If required, you can use environment-specific LoadBalancer Services or Ingress resources to expose these applications outside the cluster.

The `server` workload is suitable for traditional applications, including HTTP applications, which have the following characteristics:

- Store state locally
- Run background tasks outside of requests
- Provide multiple network ports or non-HTTP protocols
- Are not a good match for the `web` workload type

An application using the `server` workload type has the following features:

- Does not natively autoscale, but you can use these applications with the Kubernetes Horizontal Pod Autoscaler.
- By default, is exposed only within the cluster using a `ClusterIP` service.
- Uses health checks if defined by a convention.
- Uses a rolling update pattern by default.

When creating a workload with the `tanzu apps workload create` command, you can use the `--type=server` argument to select the `server` workload type. For more information, see Use the server Workload Type later in this topic. You can also use the `apps.tanzu.vmware.com/workload-type:server` annotation in the YAML workload description to support this deployment type.

Use the `server` workload type

The `spring-sensors-consumer-web` workload in Bind an application workload to the service instance in the Get started guide is a good match for the `server` workload type.

This is because it runs continuously to extract information from a RabbitMQ queue, and stores the resulting data locally in memory and presents it through a web UI.

In the Services Toolkit example in Bind an application workload to the service instance, you can update the `spring-sensors-consumer-web` workload to use the `server` supply chain by changing the workload:

```
tanzu apps workload apply spring-sensors-consumer-web --type=server
```

This shows the change in the workload label and prompts you to accept the change. After the workload finishes the new deployment, there are a few differences:

- The workload no longer exposes a URL. It's available within the cluster as `spring-sensors-consumer-web` within the namespace, but you must use `kubectl port-forward service/spring-sensors-consumer-web 8080` to access the web service on port 8080.

  You can set up a Kubernetes Ingress rule to direct traffic from outside the cluster to the workload. Use an Ingress rule to specify that specific host names or paths must be routed to the application. For more information about Ingress rules, see the Kubernetes documentation
- The workload no longer autoscales based on request traffic. For the `spring-sensors-consumer-web` workload, this means that it never spawns a second instance that consumes part of the request queue. Also, it does not scale down to zero instances.

`server`-specific workload parameters

In addition to the common supply chain parameters, `server` workloads can expose one or more network ports from the application to the Kubernetes cluster by using the `ports` parameter. This parameter is a list of port objects, similar to a Kubernetes service specification.

If you do not configure the `ports` parameter, the applied container conventions in the cluster establishes the set of exposed ports.

The following configuration exposes two ports on the Kubernetes cluster under the `my-app` host name:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: my-app
  labels:
    apps.tanzu.vmware.com/workload-type: server
spec:
  params:
  - name: ports
    value:
    - containerPort: 2025
      name: smtp
      port: 25
    - port: 8080
  ...
```

This snippet configures:

- One service on port 25, which is redirected to port 2025 on the application
- One service on port 8080, which is routed to port 8080 on the application

You can set the `ports` parameter from the `tanzu apps workload create` command as `--param-yaml 'ports=[{"port": 8080}]'`.

The following values are valid within the `ports` argument:

Field	Value
`port`	The port on which the application is exposed to the rest of the cluster
`containerPort`	The port on which the application listens for requests. Defaults to `port` if not set.
`name`	A human-readable name for the port. Defaults to `port` if not set.

Expose `server` workloads outside the cluster

You have several options for exposing `server` workloads outside the cluster:

- Expose HTTP server workloads outside the cluster manually
- Define a workload type that exposes server workloads outside the cluster
- Expose workloads outside the cluster using AVI L4/L7

Use server workloads

This topic tells you how to use the `server` workload type in Tanzu Application Platform (commonly known as TAP).

Overview

The `server` workload type allows you to deploy traditional network applications on Tanzu Application Platform.

Using an application workload specification, you can build and deploy application source code to a manually-scaled Kubernetes deployment which exposes an in-cluster Service endpoint. If required, you can use environment-specific LoadBalancer Services or Ingress resources to expose these applications outside the cluster.

The `server` workload is suitable for traditional applications, including HTTP applications, which have the following characteristics:

- Store state locally
- Run background tasks outside of requests
- Provide multiple network ports or non-HTTP protocols
- Are not a good match for the `web` workload type

An application using the `server` workload type has the following features:

- Does not natively autoscale, but you can use these applications with the Kubernetes Horizontal Pod Autoscaler.
- By default, is exposed only within the cluster using a `ClusterIP` service.
- Uses health checks if defined by a convention.
- Uses a rolling update pattern by default.

When creating a workload with the `tanzu apps workload create` command, you can use the `--type=server` argument to select the `server` workload type. For more information, see Use the server Workload Type later in this topic. You can also use the `apps.tanzu.vmware.com/workload-type:server` annotation in the YAML workload description to support this deployment type.

Use the `server` workload type

The `spring-sensors-consumer-web` workload in Bind an application workload to the service instance in the Get started guide is a good match for the `server` workload type.

This is because it runs continuously to extract information from a RabbitMQ queue, and stores the resulting data locally in memory and presents it through a web UI.

In the Services Toolkit example in Bind an application workload to the service instance, you can update the `spring-sensors-consumer-web` workload to use the `server` supply chain by changing the workload:

```
tanzu apps workload apply spring-sensors-consumer-web --type=server
```

This shows the change in the workload label and prompts you to accept the change. After the workload finishes the new deployment, there are a few differences:

- The workload no longer exposes a URL. It's available within the cluster as `spring-sensors-consumer-web` within the namespace, but you must use `kubectl port-forward service/spring-sensors-consumer-web 8080` to access the web service on port 8080.

  You can set up a Kubernetes Ingress rule to direct traffic from outside the cluster to the workload. Use an Ingress rule to specify that specific host names or paths must be routed to the application. For more information about Ingress rules, see the Kubernetes documentation
- The workload no longer autoscales based on request traffic. For the `spring-sensors-consumer-web` workload, this means that it never spawns a second instance that consumes part of the request queue. Also, it does not scale down to zero instances.

`server`-specific workload parameters

In addition to the common supply chain parameters, `server` workloads can expose one or more network ports from the application to the Kubernetes cluster by using the `ports` parameter. This parameter is a list of port objects, similar to a Kubernetes service specification.

If you do not configure the `ports` parameter, the applied container conventions in the cluster establishes the set of exposed ports.

The following configuration exposes two ports on the Kubernetes cluster under the `my-app` host name:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: my-app
  labels:
    apps.tanzu.vmware.com/workload-type: server
spec:
  params:
  - name: ports
    value:
    - containerPort: 2025
      name: smtp
      port: 25
    - port: 8080
  ...
```

This snippet configures:

- One service on port 25, which is redirected to port 2025 on the application
- One service on port 8080, which is routed to port 8080 on the application

You can set the `ports` parameter from the `tanzu apps workload create` command as `--param-yaml 'ports=[{"port": 8080}]'`.

The following values are valid within the `ports` argument:

Field	Value
`port`	The port on which the application is exposed to the rest of the cluster
`containerPort`	The port on which the application listens for requests. Defaults to `port` if not set.
`name`	A human-readable name for the port. Defaults to `port` if not set.

Expose `server` workloads outside the cluster

You have several options for exposing `server` workloads outside the cluster:

- Expose HTTP server workloads outside the cluster manually
- Define a workload type that exposes server workloads outside the cluster
- Expose workloads outside the cluster using AVI L4/L7

Expose HTTP server workloads outside the cluster manually

You can expose HTTP `server` workloads outside the cluster by creating an Ingress resource and using cert-manager to provision TLS-signed certificates. To do so:

1. Using the `spring-sensors-consumer-web` workload from Bind an application workload to the service instance as an example, create the following `Ingress`:

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
    name: spring-sensors-consumer-web
    namespace: DEVELOPER-NAMESPACE
    annotations:
      cert-manager.io/cluster-issuer: tap-ingress-selfsigned
      ingress.kubernetes.io/force-ssl-redirect: "true"
      kubernetes.io/ingress.class: contour
      kubernetes.io/tls-acme: "true"
   spec:
    tls:
      - secretName: spring-sensors-consumer-web
        hosts:
          - "spring-sensors-consumer-web.INGRESS-DOMAIN"
    rules:
      - host: "spring-sensors-consumer-web.INGRESS-DOMAIN"
        http:
          paths:
            - pathType: Prefix
              path: /
              backend:
                service:
                  name: spring-sensors-consumer-web
                  port:
                    number: 8080
   ```

   - Replace `DEVELOPER-NAMESPACE` with your developer namespace.
   - Replace `INGRESS-DOMAIN` with the domain name defined in `tap-values.yaml` during the installation.
   - Set the annotation `cert-manager.io/cluster-issuer` to the `shared.ingress_issuer` value configured during installation or leave it as `tap-ingress-selfsigned` to use the default value.
   - Update the port exposed by your `Service` resource, which is set as `8080` in the example.

2. Access the `server` workload with HTTPS:

   ```
   curl -k https://spring-sensors-consumer-web.INGRESS-DOMAIN
   ```

Define a workload type that exposes server workloads outside the cluster

Tanzu Application Platform (commonly known as TAP) allows you to create new workload types. You start by adding an `Ingress` resource to the `server-template ClusterConfigTemplate` when this new type of workload is created.

1. Delete the `Ingress` resource previously created.
2. Install the `yq` CLI on your local machine.
3. Save the existing `server-template` in a local file by running:

   ```
   kubectl get ClusterConfigTemplate server-template -o yaml > secure-server-templ
   ate.yaml
   ```

4. Extract the `.spec.ytt` field from this file and create another file by running:

   ```
   yq eval '.spec.ytt' secure-server-template.yaml > spec-ytt.yaml
   ```

5. In the next step, you add the `Ingress` resource snippet to `spec-ytt.yaml`. This step provides a sample `Ingress` resource snippet. Make the following edits before adding the `Ingress` resource snippet to `spec-ytt.yaml`:

   - Replace `INGRESS-DOMAIN` with the Ingress domain you set during the installation.
   - Set the annotation `cert-manager.io/cluster-issuer` to the `shared.ingress_issuer` value configured during installation or leave it as `tap-ingress-selfsigned` to use the default one.
   - This configuration is based on your workload service running on port `8080`.

   The `Ingress` resource snippet looks like this:

   ```
   ---
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: #@ data.values.workload.metadata.name
     annotations:
       cert-manager.io/cluster-issuer: tap-ingress-selfsigned
       ingress.kubernetes.io/force-ssl-redirect: "true"
       kubernetes.io/ingress.class: contour
       kubernetes.io/tls-acme: "true"
       kapp.k14s.io/change-rule: "upsert after upserting Services"
     labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.run/wo
   rkload-name": data.values.workload.metadata.name })
   spec:
     tls:
       - secretName: #@ data.values.workload.metadata.name
         hosts:
           - #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
     rules:
       - host: #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
         http:
           paths:
             - pathType: Prefix
               path: /
               backend:
                 service:
                   name: #@ data.values.workload.metadata.name
                   port:
                     number: 8080
   ```

6. Add the `Ingress` resource snippet to the `spec-ytt.yaml` file and save. Look for the `Service` resource, and insert the snippet before the last `#@ end`. For example:

```
# THE TOP OF THE FILE IS NOT SHOWN

---
apiVersion: v1
kind: Service
metadata:
  name: #@ data.values.workload.metadata.name
  labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.run/wo
rkload-name": data.values.workload.metadata.name })
spec:
  selector: #@ data.values.config.metadata.labels
  ports:
  #@ hasattr(data.values.params, "ports") and len(data.values.params.ports) or
assert.fail("one or more ports param must be provided.")
  #@ declared_ports = {}
  #@ if "ports" in data.values.params:
  #@   declared_ports = data.values.params.ports
  #@ else:
  #@   declared_ports = struct.encode([{ "containerPort": 8080, "port": 8080,
"name":   "http"}])
  #@ end
  #@ for p in merge_ports(declared_ports, data.values.config.spec.containers):
  - #@ p
  #@ end

# NEW INGRESS RESOURCE
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: #@ data.values.workload.metadata.name
  annotations:
    cert-manager.io/cluster-issuer: tap-ingress-selfsigned
    ingress.kubernetes.io/force-ssl-redirect: "true"
    kubernetes.io/ingress.class: contour
    kubernetes.io/tls-acme: "true"
    kapp.k14s.io/change-rule: "upsert after upserting Services"
  labels: #@ merge_labels({ "app.kubernetes.io/component": "run", "carto.run/wo
rkload-name": data.values.workload.metadata.name })
spec:
  tls:
    - secretName: #@ data.values.workload.metadata.name
      hosts:
        - #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
  rules:
    - host: #@ data.values.workload.metadata.name + ".INGRESS-DOMAIN"
      http:
        paths:
          - pathType: Prefix
            path: /
            backend:
              service:
                name: #@ data.values.workload.metadata.name
                port:
                  number: 8080
# END NEW INGRESS RESOURCE

#@ end

---
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: #@ data.values.workload.metadata.name + "-server"
  labels: #@ merge_labels({ "app.kubernetes.io/component": "config" })
data:
  delivery.yml: #@ yaml.encode(delivery())
```

7. Add the snippet to the `.spec.ytt` property in `secure-server-template.yaml`:

```
SPEC_YTT=$(cat spec-ytt.yaml) yq eval -i '.spec.ytt |= strenv(SPEC_YTT)' secure
-server-template.yaml
```

8. Change the name of the `ClusterConfigTemplate` to `secure-server-template` by running:

```
yq eval -i '.metadata.name = "secure-server-template"' secure-server-template.y
aml
```

9. Create the new `ClusterConfigTemplate` by running:

```
kubectl apply -f secure-server-template.yaml
```

10. Verify the new `ClusterConfigTemplate` is in the cluster by running:

```
kubectl get ClusterConfigTemplate
```

Expected output:

```
kubectl get ClusterConfigTemplate
NAME                     AGE
api-descriptors          82m
config-template          82m
convention-template      82m
secure-server-template   22s
server-template          82m
service-bindings         82m
worker-template          82m
```

11. Add the new workload type to the `tap-values.yaml`. The new workload type is named `secure-server` and the `cluster_config_template_name` is `secure-server-template`.

```
ootb_supply_chain_basic:
  supported_workloads:
    - type: web
      cluster_config_template_name: config-template
    - type: server
      cluster_config_template_name: server-template
    - type: worker
      cluster_config_template_name: worker-template
    - type: secure-server
      cluster_config_template_name: secure-server-template
```

12. Update your Tanzu Application Platform installation as follows:

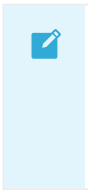
```
tanzu package installed update tap -p tap.tanzu.vmware.com --values-file \
"/path/to/your/config/tap-values.yaml"  -n tap-install
```

13. Give privileges to the `deliverable` role to manage `Ingress` resources:

```
cat <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
 name: deliverable-with-ingress
 labels:
   apps.tanzu.vmware.com/aggregate-to-deliverable: "true"
rules:
- apiGroups:
 - networking.k8s.io
 resources:
 - ingresses
 verbs:
 - get
 - list
 - watch
 - create
 - patch
 - update
 - delete
 - deletecollection
EOF
```

14. Update the workload type to `secure-server`:



Note

If you created the `Ingress` resource manually in the previous section, delete it before this.

```
tanzu apps workload apply spring-sensors-consumer-web --type=secure-server
```

15. After the process finishes, verify that the resources Deployment, Service, and Ingress appear by running:

```
kubectl get ingress,svc,deploy -l carto.run/workload-name=spring-sensors-consum
er-web
```

Expected output:

```
kubectl get ingress,svc,deploy -l carto.run/workload-name=tanzu-java-web-app-js
NAME                                                    CLASS    HOSTS
ADDRESS          PORTS     AGE
ingress.networking.k8s.io/spring-sensors-consumer-web   <none>   spring-sensors
-consumer-web.INGRESS-DOMAIN   34.111.111.111   80, 443   37s

NAME                                    TYPE        CLUSTER-IP     EXTERNAL-IP
PORT(S)    AGE
service/spring-sensors-consumer-web   ClusterIP   10.32.15.194   <none>
8080/TCP   36m

NAME                                          READY   UP-TO-DATE   AVAILABLE
AGE
deployment.apps/spring-sensors-consumer-web   1/1     1            1
37s
```

16. Access your `secure-server` workload with HTTPS by running:

```
curl -k https://spring-sensors-consumer-web.INGRESS-DOMAIN
```

Expose workloads outside the cluster using AVI L4/L7

To expose workloads outside the cluster by using AVI L4/L7, see the Tanzu Reference Architecture documentation.

Use worker workloads

This topic tells you how to create and install a supply chain for the `worker` workload type in Tanzu Application Platform (commonly known as TAP).

Overview

The `worker` workload type allows you to deploy applications that run continuously without network input on Tanzu Application Platform. Using an application workload specification, you can build and deploy application source code to a manually scaled Kubernetes deployment with no network exposure.

The `worker` workload is a good match for applications that manage their own work by reading from a worker or a background scheduled time source, and don't expose any network interfaces.

An application using the `worker` workload type has the following features:

- Does not natively auto-scale but you can use it with the Kubernetes Horizontal Pod Autoscaler
- Does not expose any network services
- Uses health checks if defined by a convention
- Uses a rolling update pattern by default

When creating a workload with `tanzu apps workload create`, you can use the `--type=worker` argument to select the `worker` workload type. For more information, see the Use the worker Workload Type section. You can also use the `apps.tanzu.vmware.com/workload-type:worker` annotation in the YAML workload description to support this deployment type.

Use the `worker` workload type

The `spring-sensors-producer` workload in the example in Consume services on Tanzu Application Platform is a good match for the `worker` workload type. This is because it runs continuously without a UI to report sensor information to a RabbitMQ topic.

If you followed the Services Toolkit example, you can update the `spring-sensors-producer` to use the `worker` supply chain by changing the workload type. To do so, run:

```
tanzu apps workload apply spring-sensors-producer --type=worker
```

This shows a difference in the workload label, and prompts you to accept the change. After the workload finishes the new deployment, there are a few differences:

- The workload no longer has a URL. Because the workload does not present a web UI, this more closely matches the original application intent.
- The workload no longer auto-scales based on request traffic. For the `spring-sensors-producer` workload, this means that it does not scale down to zero instances when there is no request traffic.

Parameter reference

This topic tells you about the default supply chains and templates provided by Tanzu Application Platform (commonly known as TAP). This topic describes the `workload.spec.params` parameters

that are configured in workload objects, and the `deliverable.spec.params` parameters that are configured in the deliverable object.

Workload Parameter Reference

The supply chains and templates provided by the Out of the Box packages contain a series of parameters that customize supply chain behavior. This section describes the `workload.spec.params` parameters that can be configured in workload objects.

The following table provides a list of supply chain resources organized by the resource in the supply chain where they are used. Some of these resources might not be applicable depending on the supply chain in use.

List of Supply Chain Resources for Workload Object

Supply Chain Resource	Output Type	Purpose	Basic	Testing	Scanning
source-provider	Source	Fetches source code	Yes	Yes	Yes
source-tester	Source	Tests source code	No	Yes	Yes
source-scanner	Source	Scans source code	No	No	Yes
image-provider	Image	Builds application container image	Yes	Yes	Yes
image-scanner	Image	Scans application container image	No	No	Yes
config-provider	Podtemplate spec	Tailors a pod spec based on the application image and conventions set up in the cluster	Yes	Yes	Yes
app-config	Kubernetes configuration	Creates Kubernetes config files (knative service/deployment - depending on workload type)	Yes	Yes	Yes
service-bindings	Kubernetes configuration	Adds service bindings to the set of config files	Yes	Yes	Yes
api-descriptors	Kubernetes configuration	Adds api descriptors to the set of config files	Yes	Yes	Yes
config-writer	Kubernetes configuration	Writes configuration to a destination (git or registry) for further deployment to a run cluster	Yes	Yes	Yes
deliverable	Kubernetes configuration	Writes deliverable content to be extracted for use in a run cluster	Yes	Yes	Yes

For information about supply chains, see:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain Testing
- Out of the Box Supply Chain Testing Scanning

source-provider

The `source-provider` resource in the supply chain creates objects that fetch either source code or pre-compiled Java applications depending on how the workload is configured. For more information, see Building from Source.

GitRepository

Use `gitrepository` when fetching source code from Git repositories. This resource makes further resources available in the supply chain, such as the contents of the Git repository as a tarball available in the cluster.

Parameters:

Parameter name	Meaning	Example
`gitImplementation`	VMware recommends that you use the underlying library for fetching the source code. Either `libgit2`, required for Azure DevOps, or `go-git`.	`- name: gitImplementation value: libgit2`
`gitops_ssh_secret`	The name of the secret in the same namespace as the `Workload` used for providing credentials for fetching source code from the Git repository. For more information, see Git authentication.	`- name: gitops_ssh_secret value: git-credentials`

It might not be necessary to change the default Git implementation, but some providers such as Azure DevOps, require you to use `libgit2` as the server-side implementation provides support only for git's v2 protocol.

For information about the features supported by each implementation, see Git implementation in the Flux documentation.

For information about how to create a workload that uses a GitHub repository as the provider of source code, see Create a workload from GitHub repository.

For more information about GitRepository objects, see Git Repository in the Flux documentation.

ImageRepository

Use the ImageRepository when fetching source code from container images. It makes the contents of the container image available as a tarball to further resources in the supply chain. The contents of the container image are fetched by using Git or Maven.

For more information, see Create a workload from local source code.

Parameters:

Parameter Name	Meaning	Example
`serviceAccount`	The name of the service account (in the same namespace as the workload) to use to provide the credentials to `ImageRepository` for fetching the container images.	`- name: serviceAccount value: default`

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For information about custom resource details, see the ImageRepository reference topic.

For information about how to use ImageRepository with the Tanzu CLI, see Create a workload.

MavenArtifact

When carrying pre-built Java artifacts, `MavenArtifact` makes the artifact available to further resources in the supply chain as a tarball. You can wrap the tarball as a container image for further deployment. Differently from `git` and `image`, its configuration is solely driven by parameters in the workload.

Parameters:

Parameter Name	Meaning	Example

`maven`	Points to the maven artifact to fetch and the polling interval.	`- name: maven`
		`value:`
		`artifactId: springboot-initial`
		`groupId: com.example`
		`version: RELEASE`
		`classifier: sources # optional`
		`type: # optional`
		`artifactRetryTimeout: 1m0s # optional`

For information about the custom resource, see the MavenArtifact reference documentation.

For information about how to use the custom resource with the `tanzu apps workload` CLI plug-in, see Create a workload from Maven repository artifact.

source-tester

The `source-tester` resource is in `ootb-supply-chain-testing` and `ootb-supply-chain-testing-scanning`. This resource is responsible for instantiating a Tekton PipelineRun object that calls the execution of a Tekton Pipeline, in the same namespace as the workload, whenever its inputs change. For example, the source code revision that you want to test changes.

A Runnable object is instantiated to ensure that there's always a run for a particular set of inputs. The parameters are passed from the workload down to Runnable's Pipeline selection mechanism through `testing_pipeline_matching_labels` and the execution of the PipelineRuns through `testing_pipeline_params`.

Parameters:

Parameter name	Meaning	Example

`testing_pipeline_matching_labels`	The set of labels to use when searching for Tekton Pipeline objects in the same namespace as the workload. By default, a Pipeline labeled as `apps.tanzu.vmware.com/pipeline: test` is selected, but when using this parameter, it's possible to override the behavior.	`- name: testing_pipeline_matching_labels`
		`value:`
		`apps.tanzu.com/pipeline: test`
		`my.company/language: golang`

Parameter name	Meaning	Example
`testing_pipeline_params`	The set of extra parameters, aside from `source-url` and `source-revision`, to pass to the Tekton Pipeline. The Tekton Pipeline **must** declare both the required parameters `source-url` and `source-revision` and the extra ones declared in this table.	`- name: testing_pipeline_params value: - name: verbose value: true`

For information about how to set up the Workload namespace for testing with Tekton, see Out of the Box Supply Chain with Testing.

For information about how to use the parameters to customize this resource to test using a Jenkins cluster, see Out of the Box Supply Chain with Testing on Jenkins.

source-scanner

The `source-scanner` resource is available in `ootb-supply-chain-testing-scanning`. It scans the source code that is tested by pointing a SourceScan object at the same source code as the tests.

You can customize behavior for both CVEs evaluation with parameters.

Parameters:

Parameter name	Meaning	Example
`scanning_source_template`	The name of the ScanTemplate object (in the same namespace as the workload) to use for running the scans against the source code.	`- name: scanning_source_template value: private-source-scan-template`
`scanning_source_policy`	The name of the ScanPolicy object (in the same namespace as the workload) to use when evaluating the scan results of a source scan.	`- name: scanning_source_policy value: allowlist-policy`

For more information, see Out of the Box Supply Chain with Testing and Scanning for details about how to set up the workload namespace with the ScanPolicy and ScanTemplate required for this resource, and SourceScan reference for details about the SourceScan custom resource.

For information about how the artifacts found during scanning are catalogued, see Supply Chain Security Tools for Tanzu – Store.

image-provider

The `image-provider` in the supply chains provides a container image carrying the application already built to further resources.

Different semantics apply, depending on how the workload is configured, for example, if using pre-built images or building from source:

- pre-built: an `ImageRepository` object is created aiming at providing a reference to the latest image found matching the name as specified in `workload.spec.image`

- building from source: an image builder object is created (either Kpack's `Image` or a `Runnable` for creating Tekton TaskRuns for building images from Dockerfiles)

Kpack Image

Use the Kpack Image object to build a container image out of source code or pre-built Java artifact.

This makes the container image available to further resources in the supply chain through a content addressable image reference that's carried to the final deployment objects unchanged. For more information, see Tanzu Build Service.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the serviceaccount (in the same namespace as the workload) to use for providing credentials to `Image` for pushing the container images it builds to the configured registry.	`- name: serviceAccount` `value: default`
`clusterBuilder`	The name of the Kpack cluster builder to use in the Kpack Image object created.	`- name: clusterBuilder` `value: nodejs-cluster-builder`

`buildServiceBindings`	The definition of a list of service bindings to use at build time. For example, providing credentials for fetching dependencies from repositories that require credentials.	`- name: buildServiceBindings`
		`value:`
		`- name: settings-xml`
		`kind: Secret`
		`apiVersion: v1`

`live-update`	Enables the use of Tilt's live-update function.	`- name: live-update` `value: "true"`

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For information about the integration with Tanzu Build Service, see Tanzu Build Service Integration.

For information about `live-update`, see Developer Conventions and Overview of Tanzu Developer Tools for IntelliJ.

For information about using Kpack builders with `clusterBuilder`, see Builders.

For information about `buildServiceBindings`, see Service Bindings.

Runnable (TaskRuns for Dockerfile-based builds)

To perform Dockerfile-based builds, all the supply chains instantiate a Runnable object that instantiates Tekton TaskRun objects to call the execution of kaniko builds.

Parameters:

Parameter name	Meaning	Example
`dockerfile`	The relative path to the Dockerfile file in the build context.	`./Dockerfile`
`docker_build_context`	The relative path to the directory where the build context is.	`.`
`docker_build_extra_args`	List of flags to pass directly to Kaniko, such as providing arguments to a build.	`- --build-arg=FOO=BAR`

For information about how to use Dockerfile-based builds and limitations associated with the function, see Dockerfile-based builds.

Pre-built image (ImageRepository)

For applications that already have their container images built outside the supply chains, such as providing an image reference under `workload.spec.image`, an `ImageRepository` object is created to keep track of any images pushed under that name.

This makes the content-addressable name, such as the image name containing the digest, available for further resources in the supply chain.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the serviceaccount (in the same namespace as the workload) to use for providing the credentials to `ImageRepository` for fetching the container images.	`- name: serviceAccount value: default`

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=...`.

For information about the ImageRepository resource, see the ImageRepository reference documentation. For information about the prebuild image function, see Using a prebuilt image.

image-scanner

The `image-scanner` resource is included only in `ootb-supply-chain-testing-scanning`.

This resource scans a container image (either built by using the supply chain or prebuilt), persisting the results in the store, and gating the image from moving forward in case the CVEs found are not compliant with the ScanPolicy referenced by the ImageScan object create for doing so.

Parameters:

Parameter name	Meaning	Example
`scanning_image_template`	The name of the ScanTemplate object (in the same namespace as the workload) to use for running the scans against a container image.	`- name: scanning_image_template value: private-image-scan-template`
`scanning_image_policy`	The name of the ScanPolicy object (in the same namespace as the workload) to use when evaluating the scan results of an image scan.	`- name: scanning_image_policy value: allowlist-policy`

For information about the ImageScan custom resource, see ImageScan reference.

For information about how the artifacts found during scanning are catalogued, see Supply Chain Security Tools for Tanzu – Store.

config-provider

The `config-provider` resource in the supply chains generates a PodTemplateSpec to use in application configs, such as Knative services and deployments, to represent the desired pod configuration to instantiate to run the application in containers. For more information, see PodTemplateSpec in the Kubernetes documentation.

The `config-provider` resource manages a PodIntent object that represents the intention of having PodTemplateSpec enhanced with conventions installed in the cluster whose final representation is then passed forward to other resources to form the final deployment configuration.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the serviceaccount (in the same namespace as the workload) to use for providing the necessary credentials to `PodIntent` for fetching the container image to inspect the metadata to pass to convention servers and the serviceAccountName set in the podtemplatespec.	`- name: serviceAccount value: default`
`annotations`	An extra set of annotations to pass down to the PodTemplateSpec.	`- name: annotations value: name: my-application version: v1.2.3 team: store`

Parameter name	Meaning	Example
`debug`	Put the workload in debug mode.	`- name: debug` `  value: "true"`
`live-update`	Enable live-updating of the code (for innerloop development).	`- name: live-update` `  value: "true"`

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For more information about the controller behind `PodIntent`, see Cartographer Conventions.

For more details about the two convention servers enabled by default in Tanzu Application Platform installations, see Developer Conventions and Spring Boot conventions.

app-config

The `app-config` resource prepares a ConfigMap with the Kubernetes configuration that is used for instantiating an application in the form of a particular workload type in a cluster.

The resource is configured in the supply chain to allow, by default, three types of workloads with the selection of which workload type to apply based on the labels set in the workload object created by the developer:

- `apps.tanzu.vmware.com/workload-type: web`
- `apps.tanzu.vmware.com/workload-type: worker`
- `apps.tanzu.vmware.com/workload-type: server`

Only the `server` workload type has the following configurable parameters:

Parameter name	Meaning	example

`ports`	The set of network ports to expose from the application to the Kubernetes cluster.	`- name: ports`
		`  value:`
		`  - containerPort: 2025`
		`  name: smtp`
		`  port: 25`

For more information about the three different types of workloads, see workload types. For a more detailed overview of the ports parameter, see server-specific Workload parameters.

service-bindings

The `service-bindings` resource adds ServiceBindings to the set of Kubernetes configuration files to promote for deployment.

Parameters:

Parameter name	Meaning	Example

`annotations`	The extra set of annotations to pass down to the ServiceBinding and ResourceClaim objects.	`- name: annotations`
		`value:`
		`name: my-application`
		`version: v1.2.3`
		`team: store`

For an example, see –service-ref in Tanzu CLI documentation.

For an overview of the function, see Consume services on Tanzu Application Platform.

api-descriptors

The `api-descriptor` resource adds an APIDescriptor to the set of Kubernetes objects to deploy. This enables API auto registration.

Parameters:

Parameter name	Meaning	Example

`annotations`	An extra set of annotations to pass down to the APIDescriptor object.	`- name: annotations`
		`value:`
		`name: my-application`
		`version: v1.2.3`
		`team: store`

`api_descriptor`	Information used to fill the state that you want of the APIDescriptor object (its spec).	`- name: api_descriptor`
		`value:`
		`type: openapi`
		`location:`
		`baseURL: http://petclinic-hard-coded.my-apps.tapdemo.vmware.com/`
		`path: "/v3/api"`
		`owner: team-petclinic`
		`system: pet-clinics`
		`description: "example"`

The workload must include the `apis.apps.tanzu.vmware.com/register-api: "true"` label to activate this function.

For more details about API auto registration, see Use API Auto Registration.

config-writer (git or registry)

The `config-writer` resource is responsible for performing the last mile of the supply chain: persisting in an external system (registry or Git repository) the Kubernetes configuration generated throughout the supply chain.

There are three methods:

- Publishing the configuration to a container image registry
- Publishing the configuration to a Git repository by using the push of a commit
- Publishing the configuration to a Git repository by pushing a commit and opening a pull request

For more information about the different modes of operation, see Gitops vs RegistryOps.

deliverable

The `deliverable` resource creates a `deliverable` object that represents the intention of delivering to the cluster the configurations that are produced by the supply chain.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the serviceaccount (in the same namespace as the deliverable) to use for providing the necessary permissions to create the children objects for deploying the objects created by the supply chain to the cluster.	`- name: serviceAccount` `value: default`

The `--service-account` flag sets the `spec.serviceAccountName` key in the workload object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

On build clusters where a corresponding `ClusterDelivery` doesn't exist, the deliverable takes no effect (similarly to a workload without a SupplyChain, no action is taken).

Deliverable Parameters Reference

The deliverable object applies the configuration produced by the resources defined by a ClusterSupplyChain to a Kubernetes cluster.

This section describes the `deliverable.spec.params` parameters that can be configured in the deliverable object. The following section describes the two resources defined in the ClusterDelivery resources section. These are part of the `ootb-delivery-basic` package:

List of Cluster Delivery Resources for Deliverable Object

Cluster Delivery Resource	Output Type	Purpose
source provider	Source	Fetches the Kubernetes configuration file from Git repository or image registry
app deployer	Source	Applies configuration produced by a supply chain to the cluster

For information about the ClusterDelivery shipped with `ootb-delivery-basic`, and the templates used by it, see:

- Out of the Box Delivery Basic
- Out of the Templates

For information about the use of the deliverable object in a multicluster environment, see Getting started with multicluster Tanzu Application Platform.

For reference information about deliverable, see Deliverable and Delivery custom resources in the Cartographer documentation.

source-provider

The `source-provider` resource in the basic ClusterDelivery creates objects that continuously fetch Kubernetes configuration files from a Git repository or container image registry so that it can apply those to the cluster.

Regardless of where it fetches that Kubernetes configuration from (Git repository or image registry), it exposes those files to further resources along the ClusterDelivery as a tarball.

GitRepository

A GitRepository object is instantiated when `deliverable.spec.source.git` is configured to continuously look for a Kubernetes configuration pushed to a Git repository, making it available for resources in the ClusterDelivery.

Parameters:

Parameter name	Meaning	Example
`gitImplementation`	VMware recommends that you use the underlying library for fetching the source code. Either `libgit2`, required for Azure DevOps, or `go-git`.	`- name: gitImplementation value: libgit2`
`gitops_ssh_secret`	The name of the secret in the same namespace as the `deliverable` used for providing credentials for fetching Kubernetes configuration files from the Git repository pointed at. See [Git authentication](../scc/git-auth.md).	`- name: gitops_ssh_secret value: git-credentials`

It might not be necessary to change the default Git implementation but some providers, such as Azure DevOps, require you to use `libgit2` as the server-side implementation provides support only for git's v2 protocol.

For information about the features supported by each implementation, see git implementation in the Flux documentation.

For information about how to create a workload that uses a GitHub repository as the provider of source code, see Create a workload from GitHub repository.

For information about GitRepository objects, see GitRepository.

ImageRepository

An ImageRepository object is instantiated when `deliverable.spec.source.image` is configured to continuously look for Kubernetes configuration files pushed to a container image registry as opposed to a Git repository.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the service account, in the same namespace as the deliverable, you want to use to provide the necessary permissions for `kapp-controller` to deploy the objects to the cluster.	`- name: serviceAccount` `value: default`

The `--service-account` flag sets the `spec.serviceAccountName` key in the deliverable object. To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For information about custom resource details, see the ImageRepository reference documentation.

app deployer

The `app-deploy` resource in the ClusterDelivery applies the Kubernetes configuration that is built by the supply chain, pushed to either a Git repository or image repository, and applied to the cluster.

App

Regardless of where the configuration comes from, an App object is instantiated to deploy the set of Kubernetes configuration files to the cluster.

Parameters:

Parameter name	Meaning	Example
`serviceAccount`	The name of the service account, in the same namespace as the deliverable, you want to use to provide the necessary privileges for `App` to apply the Kubernetes objects to the cluster.	`- name: serviceAccount` `value: default`
`gitops_sub_path` (deprecated)	The subdirectory within the configuration bundle used for looking up the files to apply to the Kubernetes cluster.	`- name: gitops_sub_path` `value: ./config`

The `gitops_sub_path` parameter is deprecated. Use `deliverable.spec.source.subPath` instead.

The `--service-account` flag sets the `spec.serviceAccountName` key in the deliverable object.

To configure the `serviceAccount` parameter, use `--param serviceAccount=SERVICE-ACCOUNT`.

For details about RBAC and how `kapp-controller` uses the ServiceAccount provided to it using the `serviceAccount` parameter in the `deliverable` object, see kapp-controller's Security Model in the Carvel documentation.

Use functions (Beta)

This topic tells you how to create and deploy a HTTP or CloudEvent function from an Application Accelerator starter template in an online or air-gapped environment on Tanzu Application Platform (commonly known as TAP).

Important

Function Buildpacks for Knative and the corresponding Application Accelerator starter templates for Python and Java are deprecated and will be removed in Tanzu Application Platform v1.7. This beta product will not receive any future updates or patches.

Overview

The function experience on Tanzu Application Platform enables you to deploy functions, use starter templates to bootstrap your function, and write only the code that matters to your business. You can run a single CLI command to deploy your functions to an auto-scaled cluster.

Functions provide a quick way to get started writing an application. Compared with a traditional application:

- Functions have a single entry-point and perform a single task. This means that functions can be easier to understand and monitor.
- The function buildpack manages the webserver. This means that you can focus on your business logic.
- A traditional webserver application might be a better fit if you want to implement an entire website or API in a single container

Important

Beta features have been tested for functionality, but not performance. Features enter the beta stage so that customers can gain early access, and give feedback on the design and behavior.

Beta features might undergo changes based on this feedback before the end of the beta stage. VMware discourages running beta features in production. VMware cannot guarantee that you can upgrade any beta feature in the future.

Supported languages and frameworks

For HTTP and CloudEvents:

Language/framework	HTTP	CloudEvents
Java	✓	✓
Python	✓	✓
NodeJS	✓	N/A

For REST API:

Language/framework	GET	POST
Java	N/A	✓
Python	✓	✓
NodeJS	✓	✓

Prerequisites

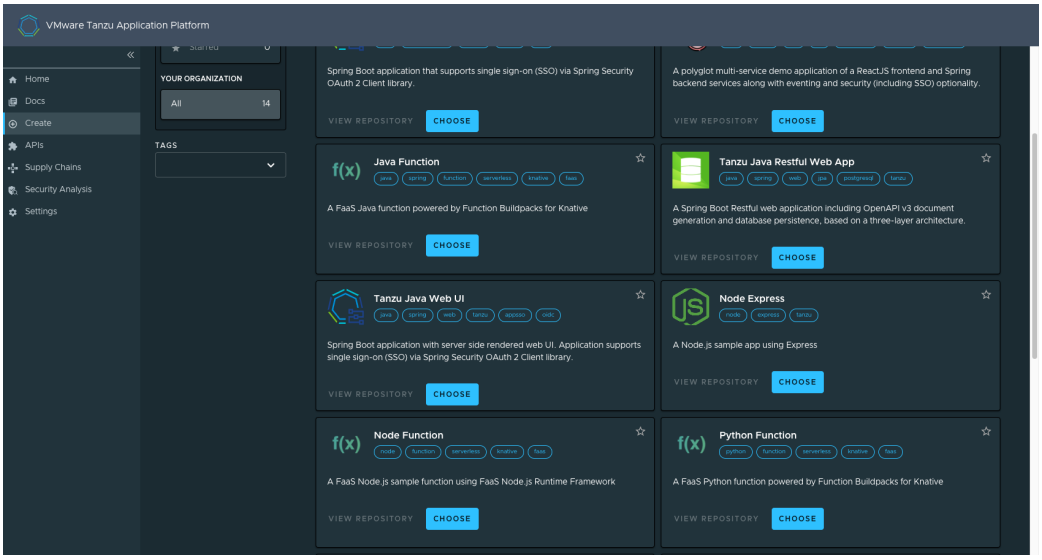
Before using function workloads, follow all instructions to install Tanzu Application Platform for your environment:

- Installing Tanzu Application Platform online
- Installing Tanzu Application Platform in an air-gapped environment

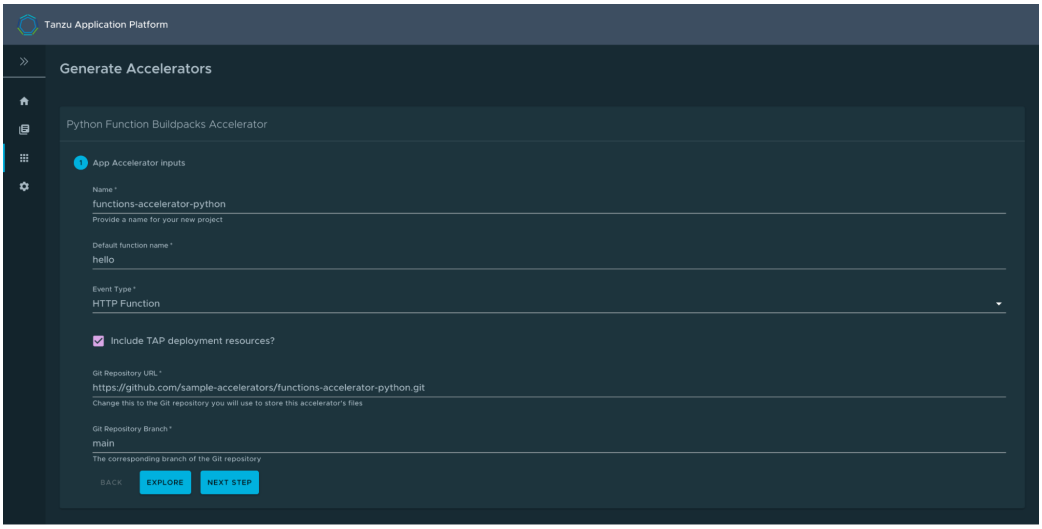
Create a function project from an accelerator

To create a function project from an accelerator:

1. From the Tanzu Application Platform GUI portal, click **Create** on the left navigation bar to see the list of available accelerators.



2. Locate the function accelerator in the language or framework of your choice and click **CHOOSE**.
3. Provide a name for your function project and your function.
4. If you are creating a Java function, select a project type.
5. Provide a Git repository to store the files for the accelerator.
6. Click **NEXT STEP**, verify the provided information, and then click **CREATE**.



7. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.

8. After downloading the ZIP file, expand it in a workspace directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Create a function project using the Tanzu CLI

From the CLI, to generate a function project using an accelerator template and then download the project artifacts as a ZIP file:

1. Verify that you have added the function accelerator template to the application accelerator server by running:

   ```
   tanzu accelerator list
   ```

2. Get the `server-url` for the Application Accelerator server. The URL depends on the configuration settings for Application Accelerator:

   - For installations configured with a shared ingress, use `https://accelerator.DOMAIN` where `DOMAIN` is provided in the values file for the accelerator configuration.
   - For installations using a LoadBalancer, look up the External IP address by running:

     ```
     kubectl get -n accelerator-system service/acc-server
     ```

     Use `http://EXTERNAL-IP` as the URL.
   - For any other configuration, you can use port forwarding by running:

     ```
     kubectl port-forward service/acc-server -n accelerator-system 8877:80
     ```

     Use `http://localhost:8877` as the URL.

3. Generate a function project from an accelerator template by running:

   ```
   tanzu accelerator generate ACCELERATOR-NAME \
   --options '{"projectName": "FUNCTION-NAME", "interfaceType": "TYPE"}' \
   --server-url APPLICATION-ACCELERATOR-URL
   ```

   Where:

   - `ACCELERATOR-NAME` is the name of the function accelerator template you want to use.
   - `FUNCTION-NAME` is the name of your function project.
   - `TYPE` is the interface you want to use for your function. Available options are `http` or `cloudevents`. CloudEvents is experimental.
   - `APPLICATION-ACCELERATOR-URL` is the URL for the Application Accelerator server that you retrieved in the previous step.

   For example:

   ```
   tanzu accelerator generate java-function \
   --options '{"projectName": "my-func", "interfaceType": "http"}' \
   --server-url http://localhost:8877
   ```

4. After generating the ZIP file, expand it in your directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Deploy your function

To deploy and verify your function:

1. Deploy the function accelerator by running the `tanzu apps workload` create command:

   ```
   tanzu apps workload create functions-accelerator-python \
   --local-path . \
   --source-image SOURCE-IMAGE \
   --type web \
   --yes
   --namespace YOUR-DEVELOPER-NAMESPACE
   --build-env 'BP_FUNCTION=func.hello'
   ```

   Where:

   - `SOURCE-IMAGE` is a writable repository in your registry in the form `REGISTRY/IMAGE:TAG`.
     - Harbor has the form: “my-harbor.io/my-project/functions-accelerator-python”.
     - Docker Hub has the form: “my-dockerhub-user/functions-accelerator-python”.
     - Google Cloud Registry has the form: “gcr.io/my-project/functions-accelerator-python”.
   - `YOUR-DEVELOPER-NAMESPACE` is the namespace you configured earlier.

2. View the build and runtime logs for your application by running the `tail` command:

   ```
   tanzu apps workload tail functions-accelerator-python --since 10m --timestamp -
   -namespace YOUR-DEVELOPER-NAMESPACE
   ```

   Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

3. After the workload is built and running, you can view the web application in your browser. To view the URL of the web application, run the following command and then **ctrl-click** the Workload Knative Services URL at the bottom of the command output.

   ```
   tanzu apps workload get functions-accelerator-python --namespace YOUR-DEVELOPER
   -NAMESPACE
   ```

   Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

4. (Optional) You can test your function using a curl command. To do so, you must have curl installed on your computer. Java function POST example:

   ```
   curl -w'\n' URL-FROM-YOUR-WORKLOAD-KNATIVE-SERVICES-SECTION \
   -H "Content-Type: application/json" \
   -d '{"firstName":"John", "lastName":"Doe"}'
   ```

   For language support for the REST API, see Supported languages and frameworks earlier in this topic.

Use functions (Beta)

This topic tells you how to create and deploy a HTTP or CloudEvent function from an Application Accelerator starter template in an online or air-gapped environment on Tanzu Application Platform (commonly known as TAP).

Important

Function Buildpacks for Knative and the corresponding Application Accelerator starter templates for Python and Java are deprecated and will be removed in Tanzu Application Platform v1.7. This beta product will not receive any future updates or patches.

Overview

The function experience on Tanzu Application Platform enables you to deploy functions, use starter templates to bootstrap your function, and write only the code that matters to your business. You can run a single CLI command to deploy your functions to an auto-scaled cluster.

Functions provide a quick way to get started writing an application. Compared with a traditional application:

- Functions have a single entry-point and perform a single task. This means that functions can be easier to understand and monitor.
- The function buildpack manages the webserver. This means that you can focus on your business logic.
- A traditional webserver application might be a better fit if you want to implement an entire website or API in a single container

Important

Beta features have been tested for functionality, but not performance. Features enter the beta stage so that customers can gain early access, and give feedback on the design and behavior.

Beta features might undergo changes based on this feedback before the end of the beta stage. VMware discourages running beta features in production. VMware cannot guarantee that you can upgrade any beta feature in the future.

Supported languages and frameworks

For HTTP and CloudEvents:

Language/framework	HTTP	CloudEvents
Java	✓	✓
Python	✓	✓
NodeJS	✓	N/A

For REST API:

Language/framework	GET	POST
Java	N/A	✓
Python	✓	✓
NodeJS	✓	✓

Prerequisites

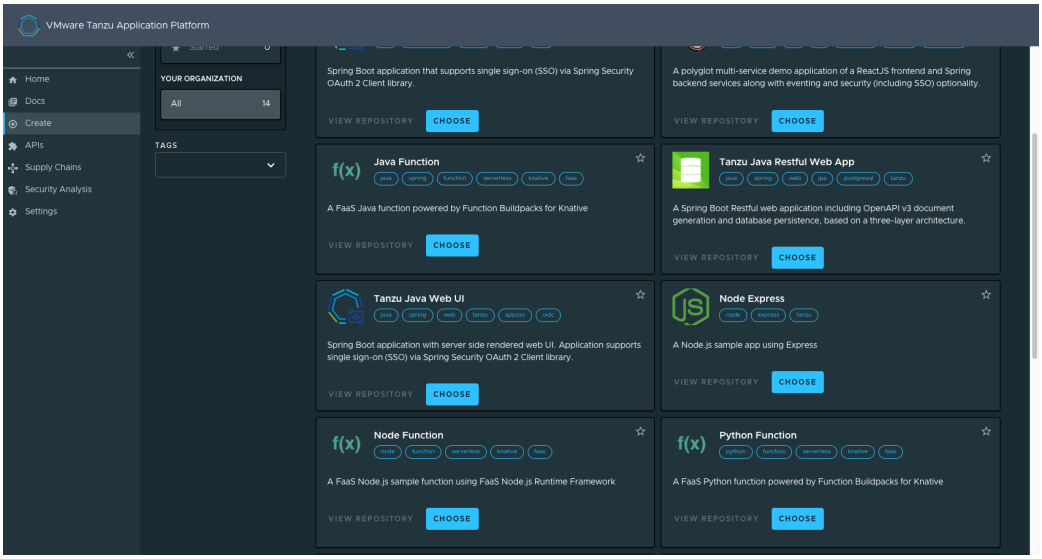
Before using function workloads, follow all instructions to install Tanzu Application Platform for your environment:

- Installing Tanzu Application Platform online
- Installing Tanzu Application Platform in an air-gapped environment

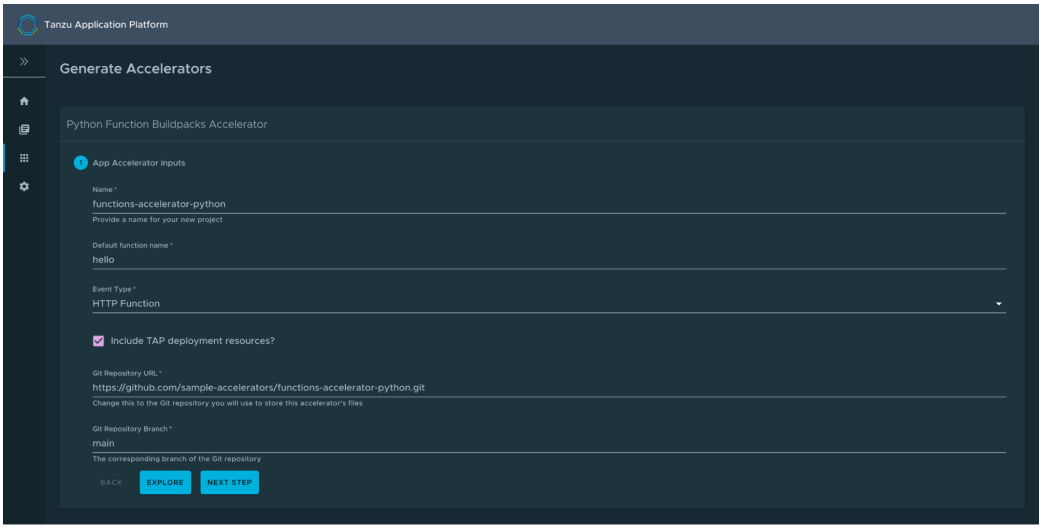
Create a function project from an accelerator

To create a function project from an accelerator:

1. From the Tanzu Application Platform GUI portal, click **Create** on the left navigation bar to see the list of available accelerators.



2. Locate the function accelerator in the language or framework of your choice and click **CHOOSE**.
3. Provide a name for your function project and your function.
4. If you are creating a Java function, select a project type.
5. Provide a Git repository to store the files for the accelerator.
6. Click **NEXT STEP**, verify the provided information, and then click **CREATE**.



7. After the Task Activity processes complete, click **DOWNLOAD ZIP FILE**.
8. After downloading the ZIP file, expand it in a workspace directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Create a function project using the Tanzu CLI

From the CLI, to generate a function project using an accelerator template and then download the project artifacts as a ZIP file:

1. Verify that you have added the function accelerator template to the application accelerator server by running:

   ```
   tanzu accelerator list
   ```

2. Get the `server-url` for the Application Accelerator server. The URL depends on the configuration settings for Application Accelerator:

   - For installations configured with a shared ingress, use `https://accelerator.DOMAIN` where `DOMAIN` is provided in the values file for the accelerator configuration.
   - For installations using a LoadBalancer, look up the External IP address by running:

     ```
     kubectl get -n accelerator-system service/acc-server
     ```

     Use `http://EXTERNAL-IP` as the URL.
   - For any other configuration, you can use port forwarding by running:

     ```
     kubectl port-forward service/acc-server -n accelerator-system 8877:80
     ```

     Use `http://localhost:8877` as the URL.

3. Generate a function project from an accelerator template by running:

   ```
   tanzu accelerator generate ACCELERATOR-NAME \
   --options '{"projectName": "FUNCTION-NAME", "interfaceType": "TYPE"}' \
   --server-url APPLICATION-ACCELERATOR-URL
   ```

   Where:

   - `ACCELERATOR-NAME` is the name of the function accelerator template you want to use.
   - `FUNCTION-NAME` is the name of your function project.
   - `TYPE` is the interface you want to use for your function. Available options are `http` or `cloudevents`. CloudEvents is experimental.
   - `APPLICATION-ACCELERATOR-URL` is the URL for the Application Accelerator server that you retrieved in the previous step.

   For example:

   ```
   tanzu accelerator generate java-function \
   --options '{"projectName": "my-func", "interfaceType": "http"}' \
   --server-url http://localhost:8877
   ```

4. After generating the ZIP file, expand it in your directory and follow your preferred procedure for uploading the generated project files to a Git repository for your new project.

Deploy your function

To deploy and verify your function:

1. Deploy the function accelerator by running the `tanzu apps workload` create command:

   ```
   tanzu apps workload create functions-accelerator-python \
   --local-path . \
   ```

```
--source-image SOURCE-IMAGE \
--type web \
--yes
--namespace YOUR-DEVELOPER-NAMESPACE
--build-env 'BP_FUNCTION=func.hello'
```

Where:

- `SOURCE-IMAGE` is a writable repository in your registry in the form `REGISTRY/IMAGE:TAG`.
  - Harbor has the form: “my-harbor.io/my-project/functions-accelerator-python”.
  - Docker Hub has the form: “my-dockerhub-user/functions-accelerator-python”.
  - Google Cloud Registry has the form: “gcr.io/my-project/functions-accelerator-python”.
- `YOUR-DEVELOPER-NAMESPACE` is the namespace you configured earlier.

2. View the build and runtime logs for your application by running the `tail` command:

```
tanzu apps workload tail functions-accelerator-python --since 10m --timestamp -
-namespace YOUR-DEVELOPER-NAMESPACE
```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

3. After the workload is built and running, you can view the web application in your browser. To view the URL of the web application, run the following command and then **ctrl-click** the Workload Knative Services URL at the bottom of the command output.

```
tanzu apps workload get functions-accelerator-python --namespace YOUR-DEVELOPER
-NAMESPACE
```

Where `YOUR-DEVELOPER-NAMESPACE` is the namespace configured earlier.

4. (Optional) You can test your function using a curl command. To do so, you must have curl installed on your computer. Java function POST example:

```
curl -w'\n' URL-FROM-YOUR-WORKLOAD-KNATIVE-SERVICES-SECTION \
-H "Content-Type: application/json" \
-d '{"firstName":"John", "lastName":"Doe"}'
```

For language support for the REST API, see Supported languages and frameworks earlier in this topic.

Troubleshoot Tanzu Application Platform

These topics provide you with troubleshooting information to help resolve issues with your Tanzu Application Platform (commonly known as TAP):

- Troubleshoot installing Tanzu Application Platform
- Troubleshoot using Tanzu Application Platform
- Troubleshoot Tanzu Application Platform components
- Troubleshoot Tanzu GitOps Reference Implementation (RI)

Troubleshoot Tanzu Application Platform

These topics provide you with troubleshooting information to help resolve issues with your Tanzu Application Platform (commonly known as TAP):

- Troubleshoot installing Tanzu Application Platform
- Troubleshoot using Tanzu Application Platform
- Troubleshoot Tanzu Application Platform components
- Troubleshoot Tanzu GitOps Reference Implementation (RI)

Troubleshoot installing Tanzu Application Platform

This topic tells you how to troubleshoot installing Tanzu Application Platform (commonly known as TAP).

Developer cannot be verified when installing Tanzu CLI on macOS

You see the following error when you run Tanzu CLI commands, for example `tanzu version`, on macOS:

```
"tanzu" cannot be opened because the developer cannot be verified
```

Explanation

Security settings are preventing installation.

Solution

To resolve this issue:

1. Click **Cancel** in the macOS prompt window.
2. Open **System Preferences** > **Security & Privacy**.
3. Click **General**.
4. Next to the warning message for the Tanzu binary, click **Allow Anyway**.

5. Enter your system username and password in the macOS prompt window to confirm the changes.
6. In the terminal window, run:

   ```
   tanzu version
   ```

7. In the macOS prompt window, click **Open**.

Access `.status.usefulErrorMessage` details

When installing Tanzu Application Platform, you receive an error message that includes the following:

```
(message: Error (see .status.usefulErrorMessage for details))
```

Explanation

A package fails to reconcile and you must access the details in `.status.usefulErrorMessage`.

Solution

Access the details in `.status.usefulErrorMessage` by running:

```
kubectl get packageinstall PACKAGE-NAME -n tap-install -o yaml
```

Where `PACKAGE-NAME` is the name of the package to target.

“Unauthorized to access” error

When running the `tanzu package install` command, you receive an error message that includes the error:

```
UNAUTHORIZED: unauthorized to access repository
```

For example:

```
$ tanzu package install app-live-view -p appliveview.tanzu.vmware.com -v 0.1.0 -n tap-
install --values-file ./app-live-view.yaml

Error: package reconciliation failed: vendir: Error: Syncing directory '0':
  Syncing directory '.' with imgpkgBundle contents:
    Imgpkg: exit status 1 (stderr: Error: Checking if image is bundle: Collecting imag
es: Working with registry.tanzu.vmware.com/app-live-view/application-live-view-install
-bundle@sha256:b13b9ba81bcc985d76607cfc04bcbb8829b4cc2820e64a99e0af840681da12aa: GET h
ttps://registry.tanzu.vmware.com/v2/app-live-view/application-live-view-install-bundl
e/manifests/sha256:b13b9ba81bcc985d76607cfc04bcbb8829b4cc2820e64a99e0af840681da12aa: U
NAUTHORIZED: unauthorized to access repository: app-live-view/application-live-view-in
stall-bundle, action: pull: unauthorized to access repository: app-live-view/applicati
on-live-view-install-bundle, action: pull
```

Note

This example shows an error received when with Application Live View as the package. This error can also occur with other packages.

Explanation

The Tanzu Network credentials needed to access the package may be missing or incorrect.

Solution

To resolve this issue:

1. Repeat the step to create a secret for the namespace. For instructions, see Add the Tanzu Application Platform Package Repository in *Installing the Tanzu Application Platform Package and Profiles*. Ensure that you provide the correct credentials.

   When the secret has the correct credentials, the authentication error should resolve itself and the reconciliation succeed. Do not reinstall the package.

2. List the status of the installed packages to confirm that the reconcile has succeeded. For instructions, see Verify the Installed Packages in *Installing Individual Packages*.

"Serviceaccounts already exists" error

When running the `tanzu package install` command, you receive the following error:

```
failed to create ServiceAccount resource: serviceaccounts already exists
```

For example:

```
$ tanzu package install app-accelerator -p accelerator.apps.tanzu.vmware.com -v 0.2.0
-n tap-install --values-file app-accelerator-values.yaml

Error: failed to create ServiceAccount resource: serviceaccounts "app-accelerator-tap-
install-sa" already exists
```

Note

This example shows an error received with App Accelerator as the package. This error can also occur with other packages.

Explanation

The `tanzu package install` command may be executed again after failing.

Solution

To update the package, run the following command after the first use of the `tanzu package install` command

```
tanzu package installed update
```

After package installation, one or more packages fails to reconcile

You run the `tanzu package install` command and one or more packages fails to install.

For example:

```
tanzu package install tap -p tap.tanzu.vmware.com -v 0.4.0 -n tap-install --values-fil
e tap-values.yaml
- Installing package 'tap.tanzu.vmware.com'
\ Getting package metadata for 'tap.tanzu.vmware.com'
| Creating service account 'tap-tap-install-sa'
/ Creating cluster admin role 'tap-tap-install-cluster-role'
| Creating cluster role binding 'tap-tap-install-cluster-rolebinding'
| Creating secret 'tap-tap-install-values'
```

```
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'tap'
/ 'PackageInstall' resource install status: Reconciling
| 'PackageInstall' resource install status: ReconcileFailed

Please consider using 'tanzu package installed update' to update the installed package
with correct settings


Error: resource reconciliation failed: kapp: Error: waiting on reconcile packageinstal
l/tap-gui (packaging.carvel.dev/v1alpha1) namespace: tap-install:
  Finished unsuccessfully (Reconcile failed:  (message: Error (see .status.usefulError
Message for details))). Reconcile failed: Error (see .status.usefulErrorMessage for de
tails)
Error: exit status 1
```

Explanation

Often, the cause is one of the following:

- Your infrastructure provider takes longer to perform tasks than the timeout value allows.
- A race-condition between components exists. For example, a package that uses `Ingress` completes before the shared Tanzu ingress controller becomes available.

The VMware Carvel tools kapp-controller continues to try in a reconciliation loop in these cases. However, if the reconciliation status is `failed` then there might be a configuration issue in the provided `tap-config.yaml` file.

Solution

1. Verify if the installation is still in progress by running:

   ```
   tanzu package installed list -A
   ```

   If the installation is still in progress, the command produces output similar to the following example, and the installation is likely to finish successfully.

   ```
   \ Retrieving installed packages...
     NAME                       PACKAGE-NAME
   PACKAGE-VERSION  STATUS               NAMESPACE
     accelerator                accelerator.apps.tanzu.vmware.com
   1.0.0            Reconcile succeeded  tap-install
     api-portal                 api-portal.tanzu.vmware.com
   1.0.6            Reconcile succeeded  tap-install
     appliveview                run.appliveview.tanzu.vmware.com
   1.0.0-build.3    Reconciling          tap-install
     appliveview-conventions    build.appliveview.tanzu.vmware.com
   1.0.0-build.3    Reconcile succeeded  tap-install
     buildservice               buildservice.tanzu.vmware.com
   1.4.0-build.1    Reconciling          tap-install
     cartographer               cartographer.tanzu.vmware.com
   0.1.0            Reconcile succeeded  tap-install
     cert-manager               cert-manager.tanzu.vmware.com
   1.5.3+tap.1      Reconcile succeeded  tap-install
     cnrs                       cnrs.tanzu.vmware.com
   1.1.0            Reconcile succeeded  tap-install
     contour                    contour.tanzu.vmware.com
   1.18.2+tap.1     Reconcile succeeded  tap-install
     conventions-controller     controller.conventions.apps.tanzu.vmware.com
   0.4.2            Reconcile succeeded  tap-install
     developer-conventions      developer-conventions.tanzu.vmware.com
   0.4.0-build1     Reconcile succeeded  tap-install
     fluxcd-source-controller   fluxcd.source.controller.tanzu.vmware.com
   0.16.0           Reconcile succeeded  tap-install
     grype                      grype.scanning.apps.tanzu.vmware.com
   ```

```
1.0.0            Reconcile succeeded  tap-install
  image-policy-webhook      image-policy-webhook.signing.apps.tanzu.vmware.com
1.0.0-beta.3     Reconcile succeeded  tap-install
  learningcenter            learningcenter.tanzu.vmware.com
0.1.0-build.6    Reconcile succeeded  tap-install
  learningcenter-workshops  workshops.learningcenter.tanzu.vmware.com
0.1.0-build.7    Reconcile succeeded  tap-install
  ootb-delivery-basic       ootb-delivery-basic.tanzu.vmware.com
0.5.1            Reconcile succeeded  tap-install
  ootb-supply-chain-basic   ootb-supply-chain-basic.tanzu.vmware.com
0.5.1            Reconcile succeeded  tap-install
  ootb-templates            ootb-templates.tanzu.vmware.com
0.5.1            Reconcile succeeded  tap-install
  scanning                  scanning.apps.tanzu.vmware.com
1.0.0            Reconcile succeeded  tap-install
  metadata-store            metadata-store.apps.tanzu.vmware.com
1.0.2            Reconcile succeeded  tap-install
  service-bindings          service-bindings.labs.vmware.com
0.6.0            Reconcile succeeded  tap-install
  services-toolkit          services-toolkit.tanzu.vmware.com
0.7.1            Reconcile succeeded  tap-install
  source-controller         controller.source.apps.tanzu.vmware.com
0.2.0            Reconcile succeeded  tap-install
  spring-boot-conventions   spring-boot-conventions.tanzu.vmware.com
0.2.0            Reconcile succeeded  tap-install
  tap                       tap.tanzu.vmware.com
0.4.0-build.12   Reconciling          tap-install
  tap-gui                   tap-gui.tanzu.vmware.com
1.0.0-rc.72      Reconcile succeeded  tap-install
  tap-telemetry             tap-telemetry.tanzu.vmware.com
0.1.0            Reconcile succeeded  tap-install
  tekton-pipelines          tekton.tanzu.vmware.com
0.30.0           Reconcile succeeded  tap-install
```

If the installation has stopped running, one or more reconciliations have likely failed, as seen in the following example:

```
NAME                        PACKAGE NAME
PACKAGE VERSION   DESCRIPTION
AGE
accelerator                 accelerator.apps.tanzu.vmware.com
1.0.1             Reconcile succeeded
109m
api-portal                  api-portal.tanzu.vmware.com
1.0.9             Reconcile succeeded
119m
appliveview                 run.appliveview.tanzu.vmware.com
1.0.2-build.2     Reconcile succeeded
109m
appliveview-conventions     build.appliveview.tanzu.vmware.com
1.0.2-build.2     Reconcile succeeded
109m
buildservice                buildservice.tanzu.vmware.com
1.5.0             Reconcile succeeded
119m
cartographer                cartographer.tanzu.vmware.com
0.2.1             Reconcile succeeded
117m
cert-manager                cert-manager.tanzu.vmware.com
1.5.3+tap.1       Reconcile succeeded
119m
cnrs                        cnrs.tanzu.vmware.com
1.1.0             Reconcile succeeded
109m
contour                     contour.tanzu.vmware.com
```

```
1.18.2+tap.1      Reconcile succeeded
117m
conventions-controller     controller.conventions.apps.tanzu.vmware.com
0.5.0             Reconcile succeeded
117m
developer-conventions      developer-conventions.tanzu.vmware.com
0.5.0             Reconcile succeeded
109m
fluxcd-source-controller   fluxcd.source.controller.tanzu.vmware.com
0.16.1            Reconcile succeeded
119m
grype                      grype.scanning.apps.tanzu.vmware.com
1.0.0             Reconcile failed: Error (see .status.usefulErrorMessage for d
etails)   109m
image-policy-webhook       image-policy-webhook.signing.apps.tanzu.vmware.com
1.0.1             Reconcile succeeded
117m
learningcenter             learningcenter.tanzu.vmware.com
0.1.0             Reconcile succeeded
109m
learningcenter-workshops   workshops.learningcenter.tanzu.vmware.com
0.1.0             Reconcile succeeded
103m
metadata-store             metadata-store.apps.tanzu.vmware.com
1.0.2             Reconcile succeeded
117m
ootb-delivery-basic        ootb-delivery-basic.tanzu.vmware.com
0.6.1             Reconcile succeeded
103m
ootb-supply-chain-basic    ootb-supply-chain-basic.tanzu.vmware.com
0.6.1             Reconcile succeeded
103m
ootb-templates             ootb-templates.tanzu.vmware.com
0.6.1             Reconcile succeeded
109m
scanning                   scanning.apps.tanzu.vmware.com
1.0.0             Reconcile succeeded
119m
service-bindings           service-bindings.labs.vmware.com
0.6.0             Reconcile succeeded
119m
services-toolkit           services-toolkit.tanzu.vmware.com
0.7.1             Reconcile succeeded
119m
source-controller          controller.source.apps.tanzu.vmware.com
0.2.0             Reconcile succeeded
119m
spring-boot-conventions    spring-boot-conventions.tanzu.vmware.com
0.3.0             Reconcile succeeded
109m
tap                        tap.tanzu.vmware.com
1.0.1             Reconcile failed: Error (see .status.usefulErrorMessage for d
etails)   119m
tap-gui                    tap-gui.tanzu.vmware.com
1.0.2             Reconcile succeeded
109m
tap-telemetry              tap-telemetry.tanzu.vmware.com
0.1.3             Reconcile succeeded
119m
tekton-pipelines           tekton.tanzu.vmware.com
0.30.0            Reconcile succeeded
119m
```

In this example, `packageinstall/grype` and `packageinstall/tap` have reconciliation errors.

2. To get more details on the possible cause of a reconciliation failure, run:

```
kubectl describe packageinstall/NAME -n tap-install
```

Where `NAME` is the name of the failing package. For this example it would be `grype`.

3. Use the displayed information to search for a relevant troubleshooting issue in this topic. If none exists, and you are unable to fix the described issue yourself, please contact support.
4. Repeat these diagnosis steps for any other packages that failed to reconcile.

Failure to accept an End User License Agreement error

You cannot access Tanzu Application Platform or one of its components from VMware Tanzu Network.

Explanation

You cannot access Tanzu Application Platform or one of its components from VMware Tanzu Network before accepting the relevant EULA in VMware Tanzu Network.

Solution

Follow the steps in Accept the End User License Agreements in *Installing the Tanzu CLI*.

Ingress is broken on Kind cluster

Your Contour installation cannot provide ingress to workloads when installed on a Kind cluster without a LoadBalancer solution. Your Kind cluster was created with port mappings, as described in the Kind install guide.

Explanation

In Tanzu Application Platform v1.5.12, the default configuration for `contour.envoy.service.type` is `LoadBalancer`. However, for the Envoy pods to be accessed by using the port mappings on your Kind cluster, the service must be of type `NodePort`.

Solution

Configure `contour.evnoy.service.type` to be `NodePort`. Then, configure `envoy.service.nodePorts.http` and `envoy.service.nodePorts.https` to the corresponding port mappings on your Kind node. Otherwise, the NodePort service is assigned random ports, which are not accessible through your Kind cluster.

Troubleshoot using Tanzu Application Platform

This topic tells you how to troubleshoot using Tanzu Application Platform (commonly known as TAP).

Use events to find possible causes

Events can highlight issues with components in a supply chain. For example, high occurrences of `StampedObjectApplied` or `ResourceOutputChanged` can indicate problems with trashing on a component.

To view the recent events for a workload, run:

```
kubectl describe workload.carto.run <workload-name> -n <workload-ns>
```

Missing build logs after creating a workload

You create a workload, but no logs appear when you run:

```
tanzu apps workload tail workload-name --since 10m --timestamp
```

Explanation

Common causes include:

- Misconfigured repository
- Misconfigured service account
- Misconfigured registry credentials

Solution

To resolve this issue, run:

```
kubectl get clusterbuilder.kpack.io -o yaml
```

```
kubectl get image.kpack.io <workload-name> -o yaml
```

```
kubectl get build.kpack.io -o yaml
```

Workload creation stops responding with “Builder default is not ready” message

You can see the “Builder default is not ready” message in two places:

1. The “Messages” section of the `tanzu apps workload get my-app` command.
2. The Supply Chain section of Tanzu Application Platform GUI.

This message indicates there is something wrong with the Builder (the component that builds the container image for your workload).

Explanation

This message is typically encountered when the core component of the Builder (`kpack`) transitions into a bad state.

Although this isn’t the only scenario where this can happen, `kpack` can transition into a bad state when Tanzu Application Platform is deployed to a local `minikube` or `kind` cluster, and especially when that `minikube` or `kind` cluster is restarted.

Solution

1. Restart `kpack` by deleting the `kpack-controller` and `kpack-webhook` pods in the `kpack` namespace. Deleting these resources triggers their recreation:
    - `kubectl delete pods --all --namespace kpack`
2. Verify status of the replacement pods:
    - `kubectl get pods --namespace kpack`
3. Verify the workload status after the new kpack pods `STATUS` are `Running`:
    - `tanzu apps workload get YOUR-WORKLOAD-NAME`

“Workload already exists” error after updating the workload

When you update the workload, you receive the following error:

```
Error: workload "default/APP-NAME" already exists
Error: exit status 1
```

Where `APP-NAME` is the name of the app.

For example, when you run:

```
tanzu apps workload create tanzu-java-web-app \
--git-repo https://github.com/dbuchko/tanzu-java-web-app \
--git-branch main \
--type web \
--label apps.tanzu.vmware.com/has-tests=true \
--yes
```

You receive the following error

```
Error: workload "default/tanzu-java-web-app" already exists
Error: exit status 1
```

Explanation

The app is running before performing a Live Update using the same app name.

Solution

To resolve this issue, either delete the app or use a different name for the app.

Workload creation fails due to authentication failure in Docker Registry

You might encounter an error message similar to the following when creating or updating a workload by using IDE or `apps` CLI plug-in:

```
Error: Writing 'index.docker.io/shaileshp2922/build-service/tanzu-java-web-app:lates
t': Error while preparing a transport to talk with the registry: Unable to create roun
d tripper: GET https://auth.ipv6.docker.com/token?scope=repository%3Ashaileshp2922%2Fb
uild-service%2Ftanzu-java-web-app%3Apush%2Cpull&service=registry.docker.io: unexpected
status code 401 Unauthorized: {"details":"incorrect username or password"}
```

Explanation

This type of error frequently occurs when the URL set for `source image` (IDE) or `--source-image` flag (`apps` CLI plug-in) is not Docker registry compliant.

Solution

1. Verify that you can authenticate directly against the Docker registry and resolve any failures by running:

   ```
   docker login -u USER-NAME
   ```

2. Verify your `--source-image` URL is compliant with Docker.

The URL in this example `index.docker.io/shaileshp2922/build-service/tanzu-java-web-app` includes nesting. Docker registry, unlike many other registry solutions, does not support nesting.

3. To resolve this issue, you must provide an unnested URL. For example, `index.docker.io/shaileshp2922/tanzu-java-web-app`

Telemetry component logs show errors fetching the “reg-creds” secret

When you view the logs of the `tap-telemetry` controller by running `kubectl logs -n tap-telemetry <tap-telemetry-controller-<hash> -f`, you see the following error:

```
"Error retrieving secret reg-creds on namespace tap-telemetry","error":"secrets \"reg-creds\" is forbidden: User \"system:serviceaccount:tap-telemetry:controller\" cannot get resource \"secrets\" in API group \"\" in the namespace \"tap-telemetry\""
```

Explanation

The `tap-telemetry` namespace misses a role that allows the controller to list secrets in the `tap-telemetry` namespace. For more information about roles, see Role and ClusterRole Kubernetes documentation.

Solution

To resolve this issue, run:

```
kubectl patch roles -n tap-telemetry tap-telemetry-controller --type='json' -p='[{"op": "add", "path": "/rules/-", "value": {"apiGroups": [""],"resources": ["secrets"],"verbs": ["get", "list", "watch"]} }]'
```

Debug convention might not apply

If you upgrade from Tanzu Application Platform v0.4, the debug convention can not apply to the app run image.

Explanation

The Tanzu Application Platform v0.4 lacks SBOM data.

Solution

Delete existing app images that were built using Tanzu Application Platform v0.4.

Execute bit not set for App Accelerator build scripts

You cannot execute a build script provided as part of an accelerator.

Explanation

Build scripts provided as part of an accelerator do not have the execute bit set when a new project is generated from the accelerator.

Solution

Explicitly set the execute bit by running the `chmod` command:

```
chmod +x BUILD-SCRIPT-NAME
```

Where `BUILD-SCRIPT-NAME` is the name of the build script.

For example, for a project generated from the “Spring PetClinic” accelerator, run:

```
chmod +x ./mvnw
```

“No live information for pod with ID” error

After deploying Tanzu Application Platform workloads, Tanzu Application Platform GUI shows a “No live information for pod with ID” error.

Explanation

The connector must discover the application instances and render the details in Tanzu Application Platform GUI.

Solution

Recreate the Application Live View connector pod by running:

```
kubectl -n app-live-view delete pods -l=name=application-live-view-connector
```

This allows the connector to discover the application instances and render the details in Tanzu Application Platform GUI.

“image-policy-webhook-service not found” error

When installing a Tanzu Application Platform profile, you receive the following error:

```
Internal error occurred: failed calling webhook "image-policy-webhook.signing.apps.tanzu.vmware.com": failed to call webhook: Post "https://image-policy-webhook-service.image-policy-system.svc:443/signing-policy-check?timeout=10s": service "image-policy-webhook-service" not found
```

Explanation

The “image-policy-webhook-service” service cannot be found.

Solution

Redeploy the `trainingPortal` resource.

“Increase your cluster resources” error

You receive an “Increase your cluster’s resources” error.

Explanation

Node pressure can be caused by an insufficient number of nodes or a lack of resources on nodes necessary to deploy the workloads.

Solution

Follow instructions from your cloud provider to scale out or scale up your cluster.

MutatingWebhookConfiguration prevents pod admission

Admission of all pods is prevented when the `image-policy-controller-manager` deployment pods do not start before the `MutatingWebhookConfiguration` is applied to the cluster.

Explanation

Pods are prevented from starting if nodes in a cluster are scaled to zero and the webhook is forced to restart at the same time as other system components. A deadlock can occur when some components expect the webhook to verify their image signatures and the webhook is not currently running.

A known rare condition during Tanzu Application Platform profiles installation can cause this. If so, you can see a message similar to one of the following in component statuses:

```
Events:
  Type     Reason            Age                   From                   Message
  ----     ------            ----                  ----                   -------
  Warning  FailedCreate      4m28s                 replicaset-controller  Error creati
ng: Internal error occurred: failed calling webhook "image-policy-webhook.signing.app
s.tanzu.vmware.com": Post "https://image-policy-webhook-service.image-policy-system.sv
c:443/signing-policy-check?timeout=10s": no endpoints available for service "image-pol
icy-webhook-service"
```

```
Events:
  Type     Reason            Age                   From                   Message
  ----     ------            ----                  ----                   -------
  Warning FailedCreate 10m replicaset-controller Error creating: Internal error occurr
ed: failed calling webhook "image-policy-webhook.signing.apps.tanzu.vmware.com": Post
"https://image-policy-webhook-service.image-policy-system.svc:443/signing-policy-chec
k?timeout=10s": service "image-policy-webhook-service" not found
```

Solution

Delete the `MutatingWebhookConfiguration` resource to resolve the deadlock and enable the system to restart. After the system is stable, restore the `MutatingWebhookConfiguration` resource to re-enable image signing enforcement.

Important

These steps temporarily deactivate signature verification in your cluster.

1. Back up `MutatingWebhookConfiguration` to a file by running:

   ```
   kubectl get MutatingWebhookConfiguration image-policy-mutating-webhook-configur
   ation -o yaml > image-policy-mutating-webhook-configuration.yaml
   ```

2. Delete `MutatingWebhookConfiguration` by running:

   ```
   kubectl delete MutatingWebhookConfiguration image-policy-mutating-webhook-confi
   guration
   ```

3. Wait until all components are up and running in your cluster, including the `image-policy-controller-manager pods` (namespace `image-policy-system`).
4. Re-apply `MutatingWebhookConfiguration` by running:

   ```
   kubectl apply -f image-policy-mutating-webhook-configuration.yaml
   ```

Priority class of webhook's pods preempts less privileged pods

When viewing the output of `kubectl get events`, you see events similar to:

```
$ kubectl get events
LAST SEEN   TYPE      REASON             OBJECT               MESSAGE
28s         Normal    Preempted          pod/testpod          Preempted by image-polic
y-system/image-policy-controller-manager-59dc669d99-frwcp on node test-node
```

Explanation

The Supply Chain Security Tools (SCST) - Sign component uses a privileged `PriorityClass` to start its pods to prevent node pressure from preempting its pods. This can cause less privileged components to have their pods preempted or evicted instead.

Solution

- **Solution 1: Reduce the number of pods deployed by the Sign component:** If your deployment of the Sign component runs more pods than necessary, scale the deployment down as follows:
  1. Create a values file named `scst-sign-values.yaml` with the following contents:

     ```
     ---
     replicas: N
     ```

     Where `N` is an integer indicating the lowest number of pods you necessary for your current cluster configuration.
  2. Apply the new configuration by running:

     ```
     tanzu package installed update image-policy-webhook \
       --package image-policy-webhook.signing.apps.tanzu.vmware.com \
       --version 1.0.0-beta.3 \
       --namespace tap-install \
       --values-file scst-sign-values.yaml
     ```

  3. Wait a few minutes for your configuration to take effect in the cluster.
- **Solution 2: Increase your cluster's resources:** Node pressure can be caused by an insufficient number of nodes or a lack of resources on nodes necessary to deploy the workloads. Follow instructions from your cloud provider to scale out or scale up your cluster.

CrashLoopBackOff from password authentication fails

SCST - Store does not start. You see the following error in the `metadata-store-app` Pod logs:

```
$ kubectl logs pod/metadata-store-app-* -n metadata-store -c metadata-store-app
...
[error] failed to initialize database, got error failed to connect to `host=metadata-s
tore-db user=metadata-store-user database=metadata-store`: server error (FATAL: passwo
rd authentication failed for user "metadata-store-user" (SQLSTATE 28P01))
```

Explanation

The database password has changed between deployments. This is not supported.

Solution

Redeploy the app either with the original database password or follow the latter steps to erase the data on the volume:

1. Deploy `metadata-store app` with kapp.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

   ```
   kubectl exec -it metadata-store-db-KUBERNETES-ID -n metadata-store /bin/bash
   ```

   Where `KUBERNETES-ID` is the ID generated by Kubernetes and appended to the pod name.
4. To delete all database data, run:

   ```
   rm -rf /var/lib/postgresql/data/*
   ```

   This is the path found in `postgres-db-deployment.yaml`.
5. Delete the `metadata-store` app with kapp.
6. Deploy the `metadata-store` app with kapp.

Password authentication fails

SCST - Store does not start. You see the following error in the `metadata-store-app` pod logs:

```
$ kubectl logs pod/metadata-store-app-* -n metadata-store -c metadata-store-app
...
[error] failed to initialize database, got error failed to connect to `host=metadata-s
tore-db user=metadata-store-user database=metadata-store`: server error (FATAL: passwo
rd authentication failed for user "metadata-store-user" (SQLSTATE 28P01))
```

Explanation

The database password has changed between deployments. This is not supported.

Solution

Redeploy the app either with the original database password or follow the latter steps to erase the data on the volume:

1. Deploy `metadata-store app` with kapp.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

```
kubectl exec -it metadata-store-db-KUBERNETES-ID -n metadata-store /bin/bash
```

Where `KUBERNETES-ID` is the ID generated by Kubernetes and appended to the pod name.

4. To delete all database data, run:

```
rm -rf /var/lib/postgresql/data/*
```

This is the path found in `postgres-db-deployment.yaml`.

5. Delete the `metadata-store` app with kapp.
6. Deploy the `metadata-store` app with kapp.

`metadata-store-db` pod fails to start

When SCST - Store is deployed, deleted, and then redeployed, the `metadata-store-db` pod fails to start if the database password changed during redeployment.

Explanation

The persistent volume used by `PostgreSQL` retains old data, even though the retention policy is set to `DELETE`.

Solution

Redeploy the app either with the original database password or follow the later steps to erase the data on the volume:

1. Deploy `metadata-store app` with kapp.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

```
kubectl exec -it metadata-store-db-KUBERNETES-ID -n metadata-store /bin/bash
```

Where `KUBERNETES-ID` is the ID generated by Kubernetes and appended to the pod name.

4. To delete all database data, run:

```
rm -rf /var/lib/postgresql/data/*
```

This is the path found in `postgres-db-deployment.yaml`.

5. Delete the `metadata-store` app with kapp.
6. Deploy the `metadata-store` app with kapp.

Missing persistent volume

After SCST - Store is deployed, `metadata-store-db` pod fails for missing volume while `postgres-db-pv-claim` pvc is in the `PENDING` state.

Explanation

The cluster where SCST - Store is deployed does not have `storageclass` defined. The provisioner of `storageclass` is responsible for creating the persistent volume after `metadata-store-db` attaches `postgres-db-pv-claim`.

Solution

1. Verify that your cluster has `storageclass` by running:

   ```
   kubectl get storageclass
   ```

2. Create a `storageclass` in your cluster before deploying SCST - Store. For example:

   ```
   # This is the storageclass that Kind uses
   kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provision
   er/master/deploy/local-path-storage.yaml

   # set the storage class as default
   kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storage
   class.kubernetes.io/is-default-class":"true"}}}'
   ```

Failure to connect Tanzu CLI to AWS EKS clusters

When using the Tanzu CLI to connect to AWS EKS clusters, you might see one of the following errors:

- `Error: Unable to connect: connection refused. Confirm kubeconfig details and try again`
- `invalid apiVersion "client.authentication.k8s.io/v1alpha1"`

Explanation

The cause is Kubernetes v1.24 dropping support for `client.authentication.k8s.io/v1alpha1`. For more information, see aws/aws-cli/issues/6920 in GitHub.

Solution

Follow these steps to update your `aws-cli` to a supported v2.7.35 or later, and update the `kubeconfig` entry for your EKS clusters:

1. Update `aws-cli` to the latest version. For more information see AWS documentation.
2. Update the `kubeconfig` entry for your EKS clusters:

   ```
   aws eks update-kubeconfig --name ${EKS_CLUSTER_NAME} --region ${REGION}
   ```

3. In a new terminal window, run a Tanzu CLI command to verify the connection issue is resolved. For example:

   ```
   tanzu apps workload list
   ```

   Expect the command to execute without error.

Invalid repository paths are propagated

When inputting `shared.image_registry.project_path`, invalid repository paths are propagated.

Explanation

The key `shared.image_registry.project_path`, which takes input as `SERVER-NAME/REPO-NAME`, cannot take “/” at the end of the string.

Solution

Do not append “/” to the end of the string.

---

x509: certificate signed by unknown authority

Explanation

Tanzu Application Platform v1.4 introduces Shared Ingress Issuer to secure ingress communication by default. The Certificate Authority for Shared Ingress Issuer is generated as self-signed. As a result, you might see one of the following errors:

- `connection refused`
- `x509: certificate signed by unknown authority`

Solution

You can choose one of the following options to mitigate the issue:

Option 1: Configure the Shared Ingress Issuer’s Certificate Authority as a trusted Certificate Authority

Important

This is the recommended option for a secure instance.

Follow these steps to trust the Shared Ingress Issuer’s Certificate Authority in Tanzu Application Platform:

1. Extract the ClusterIssuer’s Certificate Authority.

   For default installations where `ingress_issuer` is not set in `tap_values.yml`, you can extract the ClusterIssuer’s Certificate Authority from cert-manager:

   ```
   kubectl get secret tap-ingress-selfsigned-root-ca -n cert-manager -o yaml | yq
   .data | cut -d' ' -f2 | head -1 | base64 -d
   ```

   If you overrode the default `ingress_issuer` while installing Tanzu Application Platform, you must refer to your issuer’s documentation to extract your ClusterIssuer’s Certificate Authority instead of using the command above.

2. Add the certificate to the list of trusted certificate authorities by appending the certificate authority to the `shared.ca_cert_data` field in your `tap-values.yml`.

3. Reapply your configuration:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --values-fi
   le tap-values.yml -n tap-install
   ```

Option 2: Deactivate the shared ingress issuer

Important

This option is recommended for testing purposes only.

Follow these steps to deactivate TLS for Cloud Native Runtimes, AppSSO and Tanzu Application Platform GUI:

1. Set `shared.ingress_issuer` to `""` in your `tap-values.yml`:

   ```
   shared:
     ingress_issuer: ""
   ```

2. Reapply your configuration:

   ```
   tanzu package install tap -p tap.tanzu.vmware.com -v ${TAP_VERSION} --values-fi
   le tap-values.yml -n tap-install
   ```

Troubleshoot Tanzu Application Platform components

For component-level troubleshooting, see these topics:

- Troubleshoot Application Live View
- Troubleshoot Bitnami Services
- Troubleshoot Cloud Native Runtimes for Tanzu
- Troubleshoot Crossplane
- Troubleshoot Learning Center
- Troubleshoot Service Bindings
- Troubleshoot Services Toolkit
- Troubleshoot Source Controller
- Troubleshoot Spring Boot conventions
- Troubleshoot Supply Chain Security Tools - Scan
- Troubleshoot Supply Chain Security Tools - Store
- Troubleshoot Tanzu Application Platform GUI
- Troubleshoot Tanzu Build Service
- Tanzu Build Service FAQ

Troubleshoot Tanzu GitOps Reference Implementation (RI)

This topic tells you how to troubleshoot Tanzu GitOps Reference Implementation (commonly known as RI).

Tanzu Sync application error

After the Tanzu Sync application is installed in the cluster, the main resource to check is the sync app in the `tanzu-sync` namespace:

```
kubectl -n tanzu-sync get app/sync --template='{{.status.usefulErrorMessage}}'
```

Example error:

```
kapp: Error: waiting on reconcile packageinstall/tap (packaging.carvel.dev/v1alpha1) n
amespace: tap-install:
  Finished unsuccessfully (Reconcile failed:  (message: Error (see .status.usefulError
Message for details)))
```

This indicates that the resource `packageinstall/tap` in the namespace `tap-install` failed. See the following section for the solution details.

Tanzu Application Platform install error

After the Tanzu Sync application is installed in the cluster, the Tanzu Application Platform starts to install. The resource to check is the Tanzu Application Platform package install in the `tap-install` namespace:

```
kubectl -n tap-install get packageinstall/tap --template='{{.status.usefulErrorMessag
e}}'
```

Common errors

You might encounter one of the following errors:

Given data value is not declared in schema

Error: Reconciliation fails with `Given data value is not declared in schema`

```
^ Reconcile failed:  (message: ytt: Error: Overlaying data values (in following order:
tap-install/.tanzu-managed/version.yaml, additional data values):
One or more data values were invalid
====================================

Given data value is not declared in schema
tap-values.yaml:
    |
  1 | shared:
    |

    = found: shared
    = expected: a map item with the key named "tap_install" (from tap-install/.tanzu-m
anaged/schema--tap-sensitive-values.yaml:3)
```

Problem: The values files were not generated according to the expected schema.

Solution: Ensure both non-sensitive and sensitive Tanzu Application Platform values files to adhere to the schema described in configure values.

Incorrect values example:

```
shared:
  ingress_domain: example.vmware.com
```

Correct values example:

```
tap_install:
  values:
    ingress_domain: example.vmware.com
```

Uninstall your Tanzu Application Platform by using Tanzu CLI

This document tells you how to uninstall Tanzu Application Platform (commonly known as TAP) packages from your Tanzu Application Platform package repository by using Tanzu CLI.

To uninstall Tanzu Application Platform:

- Delete the Packages
- Delete the Tanzu Application Platform Package Repository
- Remove Tanzu CLI, plug-ins, and associated files
- Remove Cluster Essentials

Delete the packages

- If you installed Tanzu Application Platform through predefined profiles, delete the `tap` metadata package by running:

  ```
  tanzu package installed delete tap --namespace tap-install
  ```

- If you installed any additional packages that were not in the predefined profiles, delete the individual packages by running:

  1. List the installed packages by running:

     ```
     tanzu package installed list --namespace tap-install
     ```

  2. Remove a package by running:

     ```
     tanzu package installed delete PACKAGE-NAME --namespace tap-install
     ```

     For example:

     ```
     $ tanzu package installed delete cloud-native-runtimes --namespace tap-in
     stall
     | Uninstalling package 'cloud-native-runtimes' from namespace 'tap-instal
     l'
     / Getting package install for 'cloud-native-runtimes'
     \ Deleting package install 'cloud-native-runtimes' from namespace 'tap-in
     stall'
     \ Package uninstall status: Reconciling
     / Package uninstall status: Deleting
     | Deleting admin role 'cloud-native-runtimes-tap-install-cluster-role'
     | Deleting role binding 'cloud-native-runtimes-tap-install-cluster-rolebi
     nding'
     | Deleting secret 'cloud-native-runtimes-tap-install-values'
     / Deleting service account 'cloud-native-runtimes-tap-install-sa'

      Uninstalled package 'cloud-native-runtimes' from namespace 'tap-install'
     ```

     Where `PACKAGE-NAME` is the name of a package listed in step 1.

3. Repeat step 2 for each individual package installed.

Delete the Tanzu Application Platform package repository

To delete the Tanzu Application Platform package repository:

1. Retrieve the name of the Tanzu Application Platform package repository by running:

   ```
   tanzu package repository list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package repository list --namespace tap-install
   - Retrieving repositories...
     NAME                  REPOSITORY
   STATUS               DETAILS
     tanzu-tap-repository  registry.tanzu.vmware.com/tanzu-application-platform/ta
   p-packages:0.2.0  Reconcile succeeded
   ```

2. Remove the Tanzu Application Platform package repository by running:

   ```
   tanzu package repository delete PACKAGE-REPO-NAME --namespace tap-install
   ```

   Where `PACKAGE-REPO-NAME` is the name of the packageRepository from the earlier step.

   For example:

   ```
   $ tanzu package repository delete tanzu-tap-repository --namespace tap-install
   - Deleting package repository 'tanzu-tap-repository'...
    Deleted package repository 'tanzu-tap-repository' in namespace 'tap-install'
   ```

Remove Tanzu CLI, plug-ins, and associated files

To completely remove the Tanzu CLI, plug-ins, and associated files, run the script for your OS:

- For Linux or MacOS, run:

  ```
  #!/bin/zsh
  rm -rf $HOME/tanzu/cli        # Remove previously downloaded cli files
  sudo rm /usr/local/bin/tanzu  # Remove CLI binary (executable)
  rm -rf ~/.config/tanzu/       # current location # Remove config directory
  rm -rf ~/.tanzu/              # old location # Remove config directory
  rm -rf ~/.cache/tanzu         # remove cached catalog.yaml
  rm -rf ~/Library/Application\ Support/tanzu-cli/* # Remove plug-ins
  ```

Remove Cluster Essentials

To completely remove Cluster Essentials, see Cluster Essentials documentation.

Uninstall Tanzu Application Platform by using GitOps

This document tells you how to uninstall Tanzu Application Platform (commonly known as TAP) when installed by using GitOps.

> **Caution**
>
> Tanzu Application Platform (GitOps) is currently in beta and is intended for evaluation and test purposes only. Do not use in a production environment.

To uninstall your Tanzu Application Platform:

- Delete Tanzu Sync Application
- Delete external resources (ESO installation only)
- Remove the Tanzu CLI, plug-ins, and associated files
- Remove Cluster Essentials

Delete Tanzu Sync Application

> **Caution**
>
> - Deleting Tanzu Sync application removes all associated resources of Tanzu Application Platform on the cluster.
> - You must delete any applications that were installed manually into the `tap-install` namespace, because they might interfere with the deletion of Tanzu Application Platform.

To delete Tanzu Sync Application, run:

```
kapp delete -a tanzu-sync
```

Delete external resources (ESO installation only)

To delete external resources from AWS, run:

```
cd $HOME/REPO-NAME/clusters/CLUSTER-NAME
./tanzu-sync/aws/scripts/delete-irsa.sh
./tanzu-sync/aws/scripts/delete-policies.sh
```

Remove the Tanzu CLI, plug-ins, and associated files

To completely remove the Tanzu CLI, plug-ins, and associated files, run the following scrips for Linux or MacOS:

```
#!/bin/zsh
rm -rf $HOME/tanzu/cli         # Remove previously downloaded cli files
sudo rm /usr/local/bin/tanzu  # Remove CLI binary (executable)
rm -rf ~/.config/tanzu/       # current location # Remove config directory
rm -rf ~/.tanzu/              # old location # Remove config directory
rm -rf ~/.cache/tanzu         # remove cached catalog.yaml
rm -rf ~/Library/Application\ Support/tanzu-cli/* # Remove plug-ins
```

Remove Cluster Essentials

To completely remove Cluster Essentials, see Cluster Essentials documentation.

Component documentation for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) is a modular platform that you can enhance by installing components. Most of the Tanzu Application Platform components are documented in this section. In some cases, a component's documentation is hosted on a separate site, and you'll find a link to it in this section.

Component documentation for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) is a modular platform that you can enhance by installing components. Most of the Tanzu Application Platform components are documented in this section. In some cases, a component's documentation is hosted on a separate site, and you'll find a link to it in this section.

Overview of Tanzu CLI

This topic tells you about the Tanzu command-line interface (commonly known as Tanzu CLI).

Tanzu CLI

The Tanzu CLI is a command-line interface that connects you to Tanzu. For example, you can use the Tanzu CLI to:

- Configure the Tanzu CLI itself
- Install and manage packages
- Create and manage application workloads

Tanzu CLI Architecture

The Tanzu CLI has a pluggable architecture. Plug-ins contain CLI commands. Here are the CLI plug-ins that can be installed with Tanzu Application Platform.

- Accelerator: manage accelerator's in a Kubernetes cluster
- Apps: manage application workloads running on workload clusters
- Insight: post and query image, package, source, and vulnerability data
- Package: package management
- Secret: secret management
- Services: discover service types, service instances, and manage resource claims

Tanzu CLI Installation

You install and initialize the Tanzu CLI on a computer. The computer can be a laptop, host, or server.

To install the CLI :

- To use the Tanzu CLI with **Tanzu Application Platform,** see Installing the Tanzu CLI.
- To use the Tanzu CLI with **Tanzu Kubernetes Grid,** see Install the Tanzu CLI and Other Tools.

Tanzu CLI Command Groups

Tanzu CLI commands are organized into command groups. To view a list of available command groups, run `tanzu`. The list of command groups that you see depends on which CLI plug-ins are installed on your local machine.

Install New Plug-ins

To install a Tanzu CLI plug-in that was not automatically downloaded when running `tanzu login` or `tanzu plugin sync`, install it manually by following these steps.

1. In a terminal, run:

   ```
   tanzu plugin install PLUGIN-NAME
   ```

2. Verify that you installed the plug-in successfully by running:

   ```
   tanzu plugin list

   NAME                DESCRIPTION
   SCOPE       DISCOVERY  VERSION  STATUS
   login               Login to the platform
   Standalone  default    v0.28.1  not installed
   management-cluster  Kubernetes management-cluster operations
   Standalone  default    v0.28.1  not installed
   package             Tanzu package management
   Standalone  default    v0.28.1  installed
   pinniped-auth       Pinniped authentication operations (usually not directly in
   voked)  Standalone  default    v0.28.1  not installed
   secret              Tanzu secret management
   Standalone  default    v0.28.1  installed
   telemetry           Configure cluster-wide telemetry settings
   Standalone  default    v0.28.1  not installed
   services            Commands for working with service instances, classes and cl
   aims    Standalone             v0.5.0   installed
   accelerator         Manage accelerators in a Kubernetes cluster
   Standalone             v1.4.1   installed
   apps                Applications on Kubernetes
   Standalone             v0.10.0  installed
   insight             post & query image, package, source, and vulnerability data
   Standalone             v1.4.3   installed
   ```

Install Local Plug-ins

If your network is not connected to the Internet or you want to download and inspect the Tanzu CLI plug-in binaries before installing, follow these steps:

1. Download the plug-in `tar.gz` from the release artifacts for your distribution.
2. Extract the `tar.gz` to a location on your local machine using the extraction tool of your choice. For example, the `tar -xvf` command.
3. From that location, run:

```
tanzu plugin install all --local /PATH/TO/FILE/
```

4. Verify that you installed the plug-ins successfully by running:

```
tanzu plugin list
NAME                DESCRIPTION
SCOPE       DISCOVERY  VERSION  STATUS
login               Login to the platform
Standalone  default    v0.28.1  not installed
management-cluster  Kubernetes management-cluster operations
Standalone  default    v0.28.1  not installed
package             Tanzu package management
Standalone  default    v0.28.1  installed
pinniped-auth       Pinniped authentication operations (usually not directly in
voked)  Standalone  default    v0.28.1  not installed
secret              Tanzu secret management
Standalone  default    v0.28.1  installed
telemetry           Configure cluster-wide telemetry settings
Standalone  default    v0.28.1  not installed
services            Commands for working with service instances, classes and cl
aims    Standalone             v0.5.0   installed
accelerator         Manage accelerators in a Kubernetes cluster
Standalone             v1.4.1   installed
apps                Applications on Kubernetes
Standalone             v0.10.0  installed
insight             post & query image, package, source, and vulnerability data
Standalone             v1.4.3   installed
```

Overview of Tanzu CLI

This topic tells you about the Tanzu command-line interface (commonly known as Tanzu CLI).

Tanzu CLI

The Tanzu CLI is a command-line interface that connects you to Tanzu. For example, you can use the Tanzu CLI to:

- Configure the Tanzu CLI itself
- Install and manage packages
- Create and manage application workloads

Tanzu CLI Architecture

The Tanzu CLI has a pluggable architecture. Plug-ins contain CLI commands. Here are the CLI plug-ins that can be installed with Tanzu Application Platform.

- Accelerator: manage accelerator's in a Kubernetes cluster
- Apps: manage application workloads running on workload clusters
- Insight: post and query image, package, source, and vulnerability data
- Package: package management
- Secret: secret management
- Services: discover service types, service instances, and manage resource claims

Tanzu CLI Installation

You install and initialize the Tanzu CLI on a computer. The computer can be a laptop, host, or server.

To install the CLI :

- To use the Tanzu CLI with **Tanzu Application Platform,** see Installing the Tanzu CLI.
- To use the Tanzu CLI with **Tanzu Kubernetes Grid,** see Install the Tanzu CLI and Other Tools.

Tanzu CLI Command Groups

Tanzu CLI commands are organized into command groups. To view a list of available command groups, run `tanzu`. The list of command groups that you see depends on which CLI plug-ins are installed on your local machine.

Install New Plug-ins

To install a Tanzu CLI plug-in that was not automatically downloaded when running `tanzu login` or `tanzu plugin sync`, install it manually by following these steps.

1. In a terminal, run:

   ```
   tanzu plugin install PLUGIN-NAME
   ```

2. Verify that you installed the plug-in successfully by running:

   ```
   tanzu plugin list

   NAME                 DESCRIPTION
   SCOPE       DISCOVERY  VERSION  STATUS
   login                Login to the platform
   Standalone  default    v0.28.1  not installed
   management-cluster  Kubernetes management-cluster operations
   Standalone  default    v0.28.1  not installed
   package              Tanzu package management
   Standalone  default    v0.28.1  installed
   pinniped-auth        Pinniped authentication operations (usually not directly in
   voked)  Standalone  default    v0.28.1  not installed
   secret               Tanzu secret management
   Standalone  default    v0.28.1  installed
   telemetry            Configure cluster-wide telemetry settings
   Standalone  default    v0.28.1  not installed
   services             Commands for working with service instances, classes and cl
   aims    Standalone             v0.5.0   installed
   accelerator          Manage accelerators in a Kubernetes cluster
   Standalone             v1.4.1   installed
   apps                 Applications on Kubernetes
   Standalone             v0.10.0  installed
   insight              post & query image, package, source, and vulnerability data
   Standalone             v1.4.3   installed
   ```

Install Local Plug-ins

If your network is not connected to the Internet or you want to download and inspect the Tanzu CLI plug-in binaries before installing, follow these steps:

1. Download the plug-in `tar.gz` from the release artifacts for your distribution.
2. Extract the `tar.gz` to a location on your local machine using the extraction tool of your choice. For example, the `tar -xvf` command.

3. From that location, run:

```
tanzu plugin install all --local /PATH/TO/FILE/
```

4. Verify that you installed the plug-ins successfully by running:

```
tanzu plugin list
NAME                 DESCRIPTION
SCOPE       DISCOVERY  VERSION  STATUS
login                Login to the platform
Standalone  default    v0.28.1  not installed
management-cluster   Kubernetes management-cluster operations
Standalone  default    v0.28.1  not installed
package              Tanzu package management
Standalone  default    v0.28.1  installed
pinniped-auth        Pinniped authentication operations (usually not directly in
voked)  Standalone  default    v0.28.1  not installed
secret               Tanzu secret management
Standalone  default    v0.28.1  installed
telemetry            Configure cluster-wide telemetry settings
Standalone  default    v0.28.1  not installed
services             Commands for working with service instances, classes and cl
aims    Standalone             v0.5.0   installed
accelerator          Manage accelerators in a Kubernetes cluster
Standalone             v1.4.1   installed
apps                 Applications on Kubernetes
Standalone             v0.10.0  installed
insight              post & query image, package, source, and vulnerability data
Standalone             v1.4.3   installed
```

Overview of Tanzu CLI plug-ins

The topics in this section tell you about the following plug-ins in your Tanzu Application Platform (commonly known as TAP):

- accelerator - The Application Accelerator Tanzu CLI plug-in includes commands for developers and operators to create and use accelerators.
- apps - This Tanzu CLI plug-in provides the ability to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform components installed.
- insight - The Tanzu Insight CLI plug-in enables querying vulnerability, image, and package data.

Overview of Tanzu CLI plug-ins

The topics in this section tell you about the following plug-ins in your Tanzu Application Platform (commonly known as TAP):

- accelerator - The Application Accelerator Tanzu CLI plug-in includes commands for developers and operators to create and use accelerators.
- apps - This Tanzu CLI plug-in provides the ability to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform components installed.
- insight - The Tanzu Insight CLI plug-in enables querying vulnerability, image, and package data.

Tanzu Apps CLI overview

This topic gives you an overview of the Tanzu Apps CLI. Use the Tanzu Apps CLI to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform (commonly known as TAP) components installed.

About workloads

Tanzu Application Platform enables you to quickly build and test applications regardless of your familiarity with Kubernetes. You can turn source code into a workload that runs in a container with a URL. A workload enables you to choose application specifications, such as repository location, environment variables, and service binding.

Tanzu Application Platform can support a range of workloads, including a serverless process that starts on demand, a constellation of microservices that functions as a logical application, or a small hello-world test application.

For information about installing the Tanzu Apps CLI, see Install Apps CLI plug-in.

Tanzu Apps CLI overview

This topic gives you an overview of the Tanzu Apps CLI. Use the Tanzu Apps CLI to create, view, update, and delete application workloads on any Kubernetes cluster that has the Tanzu Application Platform (commonly known as TAP) components installed.

About workloads

Tanzu Application Platform enables you to quickly build and test applications regardless of your familiarity with Kubernetes. You can turn source code into a workload that runs in a container with a URL. A workload enables you to choose application specifications, such as repository location, environment variables, and service binding.

Tanzu Application Platform can support a range of workloads, including a serverless process that starts on demand, a constellation of microservices that functions as a logical application, or a small hello-world test application.

For information about installing the Tanzu Apps CLI, see Install Apps CLI plug-in.

Install Tanzu Apps CLI plug-in

This topic describes how to install the Apps CLI plug-in.

Prerequisites

Ensure that you installed or updated the Tanzu CLI, for more information, see Install Tanzu CLI.

Install Tanzu Apps CLI plug-in

Run:

```
tanzu plugin install apps --group vmware-tap/default:v1.5
```

Verify that the plug-in is installed correctly:

```
tanzu apps version
# sample output
v0.12.1
```

Uninstall Apps CLI plug-in

Run:

```
tanzu plugin delete apps
```

Change clusters

The Apps CLI plug-in refers to the default kubeconfig file to access a Kubernetes cluster. When you run a `tanzu apps` command, the plug-in uses the default context that is defined in that kubeconfig file (located by default at `HOME/.kube/config`).

There are two ways to change the target cluster:

1. Use `kubectl config use-context CONTENT-NAME` to change the default context. All subsequent `tanzu apps` commands target the cluster defined in the new default kubeconfig context.
2. Include the `--context CONTENT-NAME` flag when running any `tanzu apps` command.

> **Note**
>
> Any subsequent `tanzu apps` commands that do not include the `--context CONTENT-NAME` flag continue to use the default context set in the kubeconfig.

Override the default kubeconfig

There are two approaches to overriding the default kubeconfig:

1. Set the environment variable `KUBECONFIG=PATH` to change the kubeconfig the Apps CLI plug-in will reference.

   All subsequent `tanzu apps` commands reference the non-default kubeconfig assigned to the environment variable.
2. Include the `--kubeconfig path` flag when running any `tanzu apps` command.

> **Note**
>
> Any subsequent `tanzu apps` commands that do not include the `--context CONTEXT-NAME` flag continue to use the default context set in the kubeconfig.

For more information about kubeconfig, see Configure Access to Multiple Clusters in the Kubernetes documentation.

Autocompletion

The Apps CLI plug-in has auto-completion support. The plug-in supports auto-completion for commands, positional arguments, flags, and flag values. Add one of the following commands to the shell config file according to your current setup:

Bash

```
tanzu completion bash >  HOME/.tanzu/completion.bash.inc
```

Zsh

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
tanzu completion zsh > "${fpath[1]}/_tanzu"
```

Create workloads

Create workloads from one of the following sources:

- A Git repository, for example, a Git branch, Git tag, or Git commit
- An existing local project
- An image that is pulled from a registry to deploy the application.
- A Maven repository artifact.

For more information, see Create a workload.

Debug and troubleshoot workloads

Check the workload status with the `tanzu apps workload get` and `tanzu apps workload tail` commands.

Use `tanzu apps workload get` to see the workload specification, the resources attached to it, their status and any associated high-level error messages (if they exist).

Use `tanzu apps workload tail` to see testing, scanning, build, configuration, deployment, and runtime logs associated with a workload and its progression through the supply chain.

For more information about using these commands and common errors, see Debug workloads.

Create a workload

This topics tells you how to create a workload from example source code with Tanzu Application Platform (commonly known as TAP).

Prerequisites

The following prerequisites are required to use workloads with Tanzu Application Platform:

- Install kubectl.
- Install Tanzu Application Platform components on a Kubernetes cluster. See Installing Tanzu Application Platform.
- Set your kubeconfig context to the prepared cluster `kubectl config use-context CONTEXT_NAME`.
- Install Tanzu CLI. See Install or update the Tanzu CLI and plug-ins.
- Install the Apps plug-in. See the Install Apps plug-in.
- Set up developer namespaces to use your installed packages.
- For more information about the values you can provide when creating and managing the life cycle of workloads, see Workload and Supply Chain Custom Resources in the Cartographer documentation. Alternatively, run `kubectl explain workload.spec` for the Kubernetes version running on the target cluster.

Get started with an example workload

You can create a workload from a GitHub repository or local source.

Create a workload from GitHub repository

Use the flags `--git-repo`, `--git-branch`, `--git-tag`, and `--git-commit` flags to create a workload from an existing Git repository. This allows the supply chain to get the source from the given repository to deploy the application.

To create a named workload and specify a Git source code location, run:

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanzu/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.5.0 --type web
```

Respond `Y` to prompts to complete process.

Where:

- `tanzu-java-web-app` is the name of the workload.
- `--git-repo` is the location of the code to build the workload from.
- `--sub-path` is the relative path inside the repository to treat as application root.
- `--git-tag` (optional) specifies which tag in the repository to pull the code from.
- `--git-branch` (optional) specifies which branch in the repository to pull the code from.
- `--type` distinguishes the workload type.

This process can also be done with non-publicly accessible repositories. These require authentication using credentials stored in a Kubernetes secret. The supply chain is in charge of managing these credentials.

Further information on how to set it up in Out of the Box Supply Chain with private Git repos, how the supply chain manages git repositories in How it works section and how to override parameters to customize the behavior to manage them in Workload parameters section.

View the full list of supported workload configuration options by running `tanzu apps workload apply --help`.

Create a workload from local source code

Use the `--local-path` and `--source-image` flags to create a workload from an existing local project. This allows the supplychain to generate an image (carvel-imgpkg) and push it to the given registry to be used in the workload.

To create a named workload and specify where the local source code is, run:

```
tanzu apps workload create pet-clinic --local-path /path/to/my/project --source-image springio/petclinic
```

Respond `Y` to the dialog box about publishing local source code if the image must be updated.

Where:

- `pet-clinic` is the name of the workload.
- `--local-path` points to the directory where the source code is located.
- `--source-image` is the registry path where the local source code is uploaded as an image.

The cluster needs the correct credentials and access rights in order to push the source code to the image registry. More information on authentication to publish local source in supply chain local

source authentication and a deeper explanation on how the supply chain manages this authentication in the local source how it works section.

Exclude Files

When working with local source code, you can exclude files from the source code to be uploaded within the image by creating a `.tanzuignore` file at the root of the source code.

The file must contain a list of file paths to exclude from the image including the file itself and the directories must not end with the system path separator (`/` or `\`).

For more information regarding the `.tanzuignore` file see tanzuignorefile.

Create workload from an existing image

Create a workload from an existing registry image by providing the reference to that image through the `--image` flag. The supplychain references the provided registry image when the workload is deployed.

For example:

```
tanzu apps workload create petclinic-image --image springcommunity/spring-framework-pe
tclinic
```

Respond `Y` to prompts to complete process.

Where:

- `petclinic-image` is the name of the workload.
- `--image` is an existing image, pulled from a registry, that contains the source that the workload is going to use to create the application.

Check the requirements to use a pre-built image in supply chain pre-built images requirements and how to configure the workload in order to use it.

Create a workload from Maven repository artifact

Create a workload from a Maven repository artifact Source-Controller by setting its properties through the `--maven-*` flags when using the supply chain.

The Maven repository URL is set when the supply chain is created.

To create a Maven workload using the CLI provided flags, run:

```
tanzu apps workload apply my-workload \
      --maven-artifact hello-world \
      --maven-type jar
      --maven-version 0.0.1 \
      --maven-group carto.run \
      --type web -y
```

For more information about the Maven flags, see the Maven flags command reference information.

For information about how to configure the Maven artifact authentication credentials, see Maven Repository Secret.

Working with YAML files

In many cases, workload life cycles are managed through CLI commands. However, there might be cases where managing the workload through direct interactions and edits of a `yaml` file is preferred. The Apps CLI plug-in supports using `yaml` files to meet the requirements.

When a workload is managed using a `yaml` file, that file **must contain a single workload definition**.

For example, a valid file looks similar to the following example:

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: tanzu-java-web-app
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
spec:
  source:
    git:
      url: https://github.com/vmware-tanzu/application-accelerator-samples
      ref:
        tag: tap-1.4.0
    subPath: tanzu-java-web-app
```

To create a workload from a file like the earlier example:

```
tanzu apps workload create --file my-workload-file.yaml
```

Note

When flags are passed in combination with `--file my-workload-file.yaml` the flag values take precedence over the associated property or values in the YAML.

The workload YAML definition can also be passed in through stdin as follows:

```
tanzu apps workload create --file - --yes
```

The console waits for input, and the content with valid `yaml` definitions for a workload can either be written or pasted. Then click **Ctrl-D** three times to start the workload creation. This can also be done with the `workload apply` command.

To pass a workload through `stdin`, the `--yes` flag is required. If not provided, the command fails.

Another way to pass a workload with the `--file` flag is using a URL, which, as mentioned before, must contain a raw file with the workload definition.

For example:

```
tanzu apps workload apply --file https://raw.githubusercontent.com/vmware-tanzu/apps-cli-plugin/main/pkg/commands/testdata/workload.yaml
```

Bind a service to a workload

Tanzu Application Platform supports creating a workload with binding to multiple services (ServiceBinding). The cluster supply chain is in charge of provisioning those services.

The purpose of these bindings is to provide information from a service resource to an application.

- To bind a database service to a workload, run:

  ```
  tanzu apps workload apply pet-clinic --service-ref "database=services.tanzu.vmware.com/v1alpha1:MySQL:my-prod-db"
  ```

  Where:

- `pet-clinic` is the name of the workload to be updated.
- `--service-ref` references the service using the format {service-ref-name}= {apiVersion}:{kind}:{service-binding-name}.

For more information about how to bind a service to a workload, see Consume services on Tanzu Application Platform.

Next steps

You can verify workload details and status, add environment variables, export definitions, or bind services.

1. To verify a workload status and details, use `tanzu apps workload get`.

   To get workload logs, use `tanzu apps workload tail`.

   For more information, see debug workload section.

2. To add environment variables, run:

   ```
   tanzu apps workload apply pet-clinic --env foo=bar
   ```

3. To export the workload definition into Git, or to migrate to another environment, run:

   ```
   tanzu apps workload get pet-clinic --export
   ```

4. To bind a service to a workload, see the –service-ref flag.

5. To see flags available for the workload commands, run:

   ```
   tanzu apps workload -h
   tanzu apps workload get -h
   tanzu apps workload create -h
   ```

Workload Examples

This topic provides you with examples of how to use the Tanzu Apps CLI `apps workload apply` command flags.

Custom registry credentials

Either use a custom certificate on your system or pass the path to the certificate through flags.

To pass the certificate through flags, specify:

- `--registry-ca-cert`: This is the path of the self-signed certificate needed for the custom or private registry. This is also populated with a default value through the environment variable `TANZU_APPS_REGISTRY_CA_CERT`.
- `--registry-password`: Use this when the registry requires credentials to push. The value of this flag can also be specified through `TANZU_APPS_REGISTRY_PASSWORD`.
- `--registry-username`: Use with `--registry-password` to set the registry credentials. It can also be provided as the environment variable `TANZU_APPS_REGISTRY_USERNAME`.
- `--registry-token`: Set when the registry authentication is done through a token. The value of this flag can also be taken from `TANZU_APPS_REGISTRY_TOKEN` environment variable.

For example:

```
tanzu apps workload apply WORKLOAD --local-path PATH-TO-REPO -s registry.url.nip.io/PA
CKAGE/IMAGE --type web --registry-ca-cert PATH-TO-CA-CERT.nip.io.crt --registry-userna
me USERNAME --registry-password PASSWORD

Alternatively, the same command can be run as:

```console
export TANZU_APPS_REGISTRY_CA_CERT=PATH-TO-CA-CERT.nip.io.crt
export TANZU_APPS_REGISTRY_PASSWORD=USERNAME
export TANZU_APPS_REGISTRY_USERNAME=PASSWORD

tanzu apps workload apply WORKLOAD --local-path PATH-TO-REPO -s registry.url.nip.io/PA
CKAGE/IMAGE
```

–live-update and –debug

Use the `--live-update` flag to ensure that local source code changes are reflected quickly on the running workload. This is particularly valuable when iterating on features that require the workload to be deployed and running to validate.

Live update is ideally situated for running from within one of our supported IDE extensions, but it can also be utilized independently as shown in the following Spring Boot application example:

Spring Boot application example

Prerequisites: Tilt must be installed on the client.

1. Clone the repository by running:

   ```
   git clone https://github.com/vmware-tanzu/application-accelerator-samples
   ```

2. Change into the `tanzu-java-web-app` directory.

3. In `Tiltfile`, first, change the `SOURCE_IMAGE` variable to use your registry and project.

4. At the very end of the file add:

   ```
   allow_k8s_contexts('your-cluster-name')
   ```

5. Inside the directory, run:

   ```
   tanzu apps workload apply tanzu-java-web-app --live-update --local-path . -s
   gcr.io/PROJECT/tanzu-java-web-app-live-update -y
   ```

   Expected output:

   ```
   The files and directories listed in the .tanzuignore file are being excluded fr
   om the uploaded source code.
   Publishing source in "." to "gcr.io/PROJECT/tanzu-java-web-app-live-update"...
     Published source

     Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  name: tanzu-java-web-app
      6 + |  namespace: default
      7 + |spec:
      8 + |  params:
      9 + |  - name: live-update
     10 + |    value: "true"
   ```
   ```

```
  11 + |  source:
  12 + |    image: gcr.io/PROJECT/tanzu-java-web-app-live-update:latest@sha256:
3c9fd738492a23ac532a709301fcf0c9aa2a8761b2b9347bdbab52ce9404264b
  Created workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since
1h"
To get status: "tanzu apps workload get tanzu-java-web-app"
```

6. Run Tilt to deploy the workload.

```
tilt up

Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

(space) to open the browser
(s) to stream logs (--stream=true)
(t) to open legacy terminal mode (--legacy=true)
(ctrl-c) to exit
Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

Initial Build • (Tiltfile)
Loading Tiltfile at: /path/to/repo/tanzu-java-web-app/Tiltfile
Successfully loaded Tiltfile (1.500809ms)
tanzu-java-w… │
tanzu-java-w… │ Initial Build • tanzu-java-web-app
tanzu-java-w… │ WARNING: Live Update failed with unexpected error:
tanzu-java-w… │   Cannot extract live updates on this build graph structure
tanzu-java-w… │ Falling back to a full image build + deploy
tanzu-java-w… │ STEP 1/1 — Deploying
tanzu-java-w… │      Objects applied to cluster:
tanzu-java-w… │        → tanzu-java-web-app:workload
tanzu-java-w… │
tanzu-java-w… │      Step 1 - 8.87s (Deploying)
tanzu-java-w… │      DONE IN: 8.87s
tanzu-java-w… │
tanzu-java-w… │
tanzu-java-w… │ Tracking new pod rollout (tanzu-java-web-app-build-1-build-po
d):
tanzu-java-w… │      ┊ Scheduled       - (…) Pending
tanzu-java-w… │      ┊ Initialized     - (…) Pending
tanzu-java-w… │      ┊ Ready           - (…) Pending
...
```

–export

Use this flag to retrieve the workload definition with all the extraneous, cluster-specific, properties, and values removed. For example, the status and metadata text boxes like `creationTimestamp`. This allows you to apply the workload definition to a different environment without having to make significant edits.

This means that the workload definition includes only the text boxes that were specified by the developer that created it (`--export` preserves the essence of the developer's intent for portability).

For example, if you create a workload with:

```
tanzu apps workload apply rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq-
sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:examp
le-rabbitmq-cluster-1" -t web
```

When querying the workload with `--export`, the default export format in YAML is as follows:

```
# with yaml format
    tanzu apps workload get rmq-sample-app --export
    ---
    apiVersion: carto.run/v1alpha1
    kind: Workload
    metadata:
    labels:
        apps.tanzu.vmware.com/workload-type: web
    name: rmq-sample-app
    namespace: default
    spec:
    serviceClaims:
    - name: rmq
        ref:
        apiVersion: rabbitmq.com/v1beta1
        kind: RabbitmqCluster
        name: example-rabbitmq-cluster-1
    source:
        git:
        ref:
            branch: main
        url: https://github.com/jhvhs/rabbitmq-sample

# with json format
    tanzu apps workload get rmq-sample-app --export --output json
    {
        "apiVersion": "carto.run/v1alpha1",
        "kind": "Workload",
        "metadata": {
            "labels": {
                "apps.tanzu.vmware.com/workload-type": "web"
            },
            "name": "rmq-sample-app",
            "namespace": "default"
        },
        "spec": {
            "serviceClaims": [
                {
                    "name": "rmq",
                    "ref": {
                        "apiVersion": "rabbitmq.com/v1beta1",
                        "kind": "RabbitmqCluster",
                        "name": "example-rabbitmq-cluster-1"
                    }
                }
            ],
            "source": {
                "git": {
                    "ref": {
                        "branch": "main"
                    },
                    "url": "https://github.com/jhvhs/rabbitmq-sample"
                }
            }
        }
    }
```

–output

Use this flag to retrieve the workload including all the cluster-specifics. The `--output` flag can also be used in conjunction with the `--export` flag to set the export format as `json`, `yaml`, or `yml`.

```
# with json format
tanzu apps workload get rmq-sample-app --output json # can also be used as tanzu apps
workload get rmq-sample-app -ojson
    {
        "kind": "Workload",
        "apiVersion": "carto.run/v1alpha1",
        "metadata": {
            "name": "rmq-sample-app",
            "namespace": "default",
            "uid": "3619ff6d-9e73-473a-9112-891a6d8aee9e",
            "resourceVersion": "11657434",
            "generation": 2,
            "creationTimestamp": "2022-11-28T05:10:32Z",
            "labels": {
                "apps.tanzu.vmware.com/workload-type": "web"
            },
            "managedFields": [
                {
                    "manager": "v0.10.0+dev-002cc44e",
                    "operation": "Update",
                    "apiVersion": "carto.run/v1alpha1",
                    "time": "2022-11-28T05:10:32Z",
                    "fieldsType": "FieldsV1",
                    "fieldsV1": {
                        "f:metadata": {
                            "f:labels": {
                                ".": {},
                                "f:apps.tanzu.vmware.com/workload-type": {}
                            }
                        },
                        ...
                    }
                },
                ...
            ]
        },
        ...
            "status": {
            "observedGeneration": 2,
            "conditions": [
                {
                    "type": "SupplyChainReady",
                    "status": "True",
                    "lastTransitionTime": "2022-11-28T05:10:32Z",
                    "reason": "Ready",
                    "message": ""
                },
                {
                    "type": "ResourcesSubmitted",
                    "status": "True",
                    "lastTransitionTime": "2022-11-28T05:13:33Z",
                    "reason": "ResourceSubmissionComplete",
                    "message": ""
                },
                ...
            ],
            "supplyChainRef": {
                "kind": "ClusterSupplyChain",
                "name": "source-to-url"
            },
            "resources": [
                {
                    "name": "source-provider",
                    "stampedRef": {
                        "kind": "GitRepository",
                        "namespace": "default",
```

```
                        "name": "rmq-sample-app",
                        "apiVersion": "source.toolkit.fluxcd.io/v1beta1",
                        "resource": "gitrepositories.source.toolkit.fluxcd.io"
                    },
                    "templateRef": {
                        "kind": "ClusterSourceTemplate",
                        "name": "source-template",
                        "apiVersion": "carto.run/v1alpha1"
                    },
                ...
                }
            ...
            ]
            ...
        }
        ...
    }

## with yaml format
tanzu apps workload get rmq-sample-app --output yaml # can also be used as tanzu apps
workload get rmq-sample-app -oyaml
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2022-11-28T05:10:32Z"
  generation: 2
  labels:
    apps.tanzu.vmware.com/workload-type: web
  managedFields:
  - apiVersion: carto.run/v1alpha1
      ...
    manager: v0.10.0+dev-002cc44e
    operation: Update
    time: "2022-11-28T05:10:32Z"
  - apiVersion: carto.run/v1alpha1
    fieldsType: FieldsV1
    ...
    manager: cartographer
    operation: Update
    subresource: status
    time: "2022-11-28T05:10:36Z"
  name: rmq-sample-app
  namespace: default
  resourceVersion: "11657434"
  uid: 3619ff6d-9e73-473a-9112-891a6d8aee9e
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
      url: https://github.com/jhvhs/rabbitmq-sample
status:
  conditions:
  - lastTransitionTime: "2022-11-28T05:10:32Z"
    message: ""
    reason: Ready
    status: "True"
    type: SupplyChainReady
  ...
```

```
  observedGeneration: 2
  resources:
  ...
    name: source-provider
    outputs:
    - digest: sha256:97b2cb779b4ea31339595cd204a3fec0053805eeacbbd6d6dd23af7d3000a6ae
      lastTransitionTime: "2022-11-28T05:16:01Z"
      name: url
      preview: |
        http://fluxcd-source-controller.flux-system.svc.cluster.local./gitrepository/d
efault/rmq-sample-app/73c6311eefbf724fee9ad6f4524fa24ec842ff34.tar.gz
    - digest: sha256:e7884b071fe1bbb2551d42a171043d061a7591e744705572136e689c2a154b7a
      lastTransitionTime: "2022-11-28T05:16:01Z"
      name: revision
      preview: |
        HEAD/73c6311eefbf724fee9ad6f4524fa24ec842ff34
    stampedRef:
      apiVersion: source.toolkit.fluxcd.io/v1beta1
      kind: GitRepository
      name: rmq-sample-app
      namespace: default
      resource: gitrepositories.source.toolkit.fluxcd.io
    templateRef:
      apiVersion: carto.run/v1alpha1
      kind: ClusterSourceTemplate
      name: source-template
  - conditions:
    - lastTransitionTime: "2022-11-28T05:13:25Z"
      message: ""
      reason: ResourceSubmissionComplete
      status: "True"
      type: ResourceSubmitted
    ...
    inputs:
    - name: source-provider
```

–sub-path

Use this flag to support use cases where more than one application is in a single project or repository.

Use `--sub-path` when creating a workload from a Git repository.

```
```console
tanzu apps workload apply subpathtester --git-repo https://github.com/PATH-TO-REPO --g
it-branch main --type web --sub-path SUBPATH

  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    apps.tanzu.vmware.com/workload-type: web
    7 + |  name: subpathtester
    8 + |  namespace: default
    9 + |spec:
   10 + |  source:
   11 + |    git:
   12 + |      ref:
   13 + |        branch: main
   14 + |      url: https://github.com/path-to-repo/PATH-TO-REPO
   15 + |    subPath: SUBPATH
```
```

```
  Do you want to create this workload? [yN]:
```

Use `--sub-path` when you create a workload from local source code. In the directory of the project you want to create the workload from:

```
  ```console
  tanzu apps workload apply WORKLOAD --local-path . -s gcr.io/REGISTRY/WORKLOAD-IMAGE
--sub-path SUBPATH

    Publish source in "." to "gcr.io/REGISTRY/WORKLOAD-IMAGE"? It might be visible to
others who can pull images from that repository Yes
  Publishing source in "." to "gcr.io/REGISTRY/WORKLOAD-IMAGE"...
    Published source

    Create workload:
        1 + |---
        2 + |apiVersion: carto.run/v1alpha1
        3 + |kind: Workload
        4 + |metadata:
        5 + |  name: WORKLOAD
        6 + |  namespace: default
        7 + |spec:
        8 + |  source:
        9 + |    image: gcr.io/REGISTRY/my-workload-image:latest@sha256:f28c5fedd0e902
800e6df9605ce5e20a8e835df9e87b1a0aa256666ea179fc3f
       10 + |    subPath: SUBPATH
    Do you want to create this workload? [yN]:

  ```
```

Note In cases where a workload must be created from local source code, to reduce the total amount of code that is uploaded, set the `--local-path` value to point directly to the directory containing the code rather than using `--sub-path`.

.tanzuignore file

There are many files and directories in projects that are not connected to running code (these files are not part of the final running container). When creating a workload from local source code, list these unused files and directories in the `.tanzuignore` file to avoid unnecessary consumption of resources when uploading the source.

When iterating on code with the `--live-update` flag enabled, changes to directories or files listed in `.tanzuignore` do not trigger the automatic re-deployment of the source code.

The following are some guidelines for the `.tanzuignore` file:

- The `.tanzuignore` file should include a reference to itself, as it provides no value when deployed.
- Directories must not end with the system separator `/`, or `\`.
- Comments using hashtag `#` can be included.
- If the `.tanzuignore` file contains files or directories that are not found in the source code, they are ignored.

Example of a .tanzuignore file

```
    .tanzuignore # must contain itself in order to be ignored
    # This is a comment
    this/is/a/folder/to/exclude
```

```
    this-is-a-file.ext
```

–dry-run

Use the `--dry-run` flag to prepare all the steps to submit a workload to the cluster but stop before sending it, and display an output of the final structure of the workload.

For example, when applying a workload from Git source:

```
tanzu apps workload apply rmq-app --git-repo https://github.com/jhvhs/rabbitmq-sample
--git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:example-rabb
itmq-cluster-1" -t web --dry-run
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: null
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-app
  namespace: default
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
      url: https://github.com/jhvhs/rabbitmq-sample
status:
  supplyChainRef: {}
```

Certify how a workload is created or updated in the cluster based on the current specifications passed through `--file workload.yaml` or command flags.

If there is an error applying the workload, this is shown with the `--dry-run` flag:

```
tanzu apps workload create rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq
-sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:exam
ple-rabbitmq-cluster-1" -t web --dry-run
Error: workload "default/rmq-sample-app" already exists
```

–update-strategy

Use this flag to control whether configuration properties and values passed through `--file workload.yaml` for an existing workload `merge` with, or `replace` (overwrite), existing on-cluster properties or values set for a workload.

The `--update-strategy` flag accepts two values: `merge` (default), and `replace`.

With the default `merge`:

If the `--file workload.yaml` deletes an existing on-cluster property or value, that property is not removed from the on-cluster definition. If the `--file workload.yaml` includes a new property or value, it is added to the on-cluster workload properties/values. If the `--file workload.yaml` updates an existing value for a property, that property's value on-cluster is updated.

With `replace`:

The on-cluster workload is updated to exactly what is specified in the `--file workload.yaml` definition.

The intent of the current default merge strategy is to prevent unintentional deletions of critical properties from existing workloads.

Note

The default value for the `--update-strategy flag` will change from merge to replace in Tanzu Application Platform v1.7.0.

Examples of the outcomes of both `merge` and `replace` update strategies are provided in the following examples:

- ```
  # Export workload if there is no previous yaml definition
  tanzu apps workload get spring-petclinic --export > spring-petclinic.yaml

  # modify the workload definition
  vi rmq-sample-app.yaml
  ---
  apiVersion: carto.run/v1alpha1
  kind: Workload
  metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: spring-petclinic
    apps.tanzu.vmware.com/workload-type: web
  spec:
  resources:
    requests:
      memory: 1Gi
    limits:           # delete this line
      memory: 1Gi     # delete this line
      cpu: 500m       # delete this line
  source:
    git:
      url: https://github.com/sample-accelerators/spring-petclinic
      ref:
        tag: tap-1.1
  ```

After saving the file, to verify how both of the update strategy options behave, run:

```
tanzu apps workload apply -f ./spring-petclinic.yaml --update-strategy merge # if flag
is not specified, merge is taken as default
```

This produces the following output:

```
  WARNING: Configuration file update strategy is changing. By default, provided config
uration files
will replace rather than merge existing configuration. The change will take place in t
he January 2024
Tanzu Application Platform release (use "--update-strategy" to control strategy explic
itly).

Workload is unchanged, skipping update
```

By contrast, use `replace` as follows:

```
tanzu apps workload apply -f ./spring-petclinic.yaml --update-strategy replace
```
```

This produces the following output:

```
  WARNING: Configuration file update strategy is changing. By default, provided config
uration files
will replace rather than merge existing configuration. The change will take place in t
he January 2024
Tanzu Application Platform release (use "--update-strategy" to control strategy explic
itly).

  Update workload:
...
  8,  8   |  name: spring-petclinic
  9,  9   |  namespace: default
 10, 10   |spec:
 11, 11   |  resources:
 12     - |    limits:
 13     - |      cpu: 500m
 14     - |      memory: 1Gi
 15, 12   |    requests:
 16, 13   |      memory: 1Gi
 17, 14   |  source:
 18, 15   |    git:
...
  Really update the workload "spring-petclinic"? [yN]:
```

The lines that were deleted in the YAML file are deleted as well in the workload running in the cluster. The only text boxes that remain exactly as they were created are the system populated metadata text boxes (`resourceVersion`, `uuid`, `generation`, `creationTimestamp`, `deletionTimestamp`).

Output workload after create/apply

`tanzu apps workload create/apply` commands can be used with `--output` flag which prints the workload once the process of its creation or update happens.

Since the usage of this flag is mainly for scripting processes, all the prompts can be skipped with the usage of `--yes` flag as follows:

```
tanzu apps workload apply rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq-
sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:examp
le-rabbitmq-cluster-1" --type web --output yaml --yes
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2023-04-04T16:15:41Z"
  generation: 1
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-sample-app
  namespace: default
  resourceVersion: "184277312"
  uid: faf6e581-a2a3-47ab-b2d3-4160513c72df
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
```

```
      url: https://github.com/jhvhs/rabbitmq-sample
status:
  supplyChainRef: {}
```

If it is not used with `--yes` flag, all the prompts will be printed and the workload definition will be shown at the end.

```
tanzu apps workload apply rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq-
sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:examp
le-rabbitmq-cluster-1" --type web --output yaml
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: rmq-sample-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  serviceClaims:
     11 + |  - name: rmq
     12 + |    ref:
     13 + |      apiVersion: rabbitmq.com/v1beta1
     14 + |      kind: RabbitmqCluster
     15 + |      name: example-rabbitmq-cluster-1
     16 + |  source:
     17 + |    git:
     18 + |      ref:
     19 + |        branch: main
     20 + |      url: https://github.com/jhvhs/rabbitmq-sample
  Do you want to create this workload? [yN]: y
  Created workload "rmq-sample-app"

To see logs:   "tanzu apps workload tail rmq-sample-app --timestamp --since 1h"
To get status: "tanzu apps workload get rmq-sample-app"

---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2023-04-04T15:18:13Z"
  generation: 1
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-sample-app
  namespace: default
  resourceVersion: "184169566"
  uid: 6588d398-b803-47e3-b31a-23d9a1a633a9
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
      url: https://github.com/jhvhs/rabbitmq-sample
status:
  supplyChainRef: {}
```

This flag can also be used with `--wait` or `--tail` if the intention is to retrieve the workload with everything and its status. The behavior is the same regarding the prompts: if `--yes` flag is not used, then the workload definition and the surveys are displayed and it remains waiting until the workload is in status `ready`. Otherwise, it does not show anything until the workload is ready in the cluster.

It should be made clear that, if `--tail` is used, its logs are not going to be suppressed despite the usage of `--yes` flag.

```
tanzu apps workload apply rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq-
sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:examp
le-rabbitmq-cluster-1" --type web --output yaml --yes --wait
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2023-04-04T16:22:29Z"
  generation: 1
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-sample-app
  namespace: default
  resourceVersion: "184296857"
  uid: 7fa58a71-0b41-4975-b816-781b87d02cde
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
      url: https://github.com/jhvhs/rabbitmq-sample
status:
  conditions:
  ...
  - lastTransitionTime: "2023-04-04T16:26:03Z"
    message: ""
    reason: Ready
    status: "True"
    type: Ready
  observedGeneration: 1
  resources:
  - conditions:
    - lastTransitionTime: "2023-04-04T16:22:36Z"
      message: ""
      reason: ResourceSubmissionComplete
      status: "True"
      type: ResourceSubmitted
      ...
    name: source-provider
    outputs:
    - digest: sha256:1982253401b1be2236786e3da433216d36d289d0b158fbc9ca6477ac94879e60
      lastTransitionTime: "2023-04-04T16:22:36Z"
      name: url
      preview: |
        http://fluxcd-source-controller.flux-system.svc.cluster.local./gitrepository/d
efault/rmq-sample-app/103fde37882b5510e9b3974e5fe209161b54f675.tar.gz
    ...
    stampedRef:
      apiVersion: source.toolkit.fluxcd.io/v1beta1
      kind: GitRepository
      name: rmq-sample-app
```

```
      namespace: default
      resource: gitrepositories.source.toolkit.fluxcd.io
    templateRef:
      apiVersion: carto.run/v1alpha1
      kind: ClusterSourceTemplate
      name: source-template
  - conditions:
    ...
    - lastTransitionTime: "2023-04-04T16:25:45Z"
      message: ""
      reason: Ready
      status: "True"
      type: Ready
    inputs:
    - name: source-provider
    ...
  - conditions:
    ...
    - lastTransitionTime: "2023-04-04T16:25:52Z"
      message: ""
      reason: Ready
      status: "True"
      type: Ready
    inputs:
    - name: image-provider
    name: config-provider
    outputs:
    - digest: sha256:0549f3f3fe5ef817af62ae6357465e6df1a6c901e5a7abc17468ee3f3e16c1a1
      lastTransitionTime: "2023-04-04T16:25:52Z"
      name: config
      preview: |-
        metadata:
            annotations:
                boot.spring.io/version: 2.4.9
                conventions.carto.run/applied-conventions: |-
                    spring-boot-convention/auto-configure-actuators-check
                    ...
                developer.conventions/target-containers: workload
                services.conventions.carto.run/rabbitmq: amqp-client/5.10.0
            labels:
                app.kubernetes.io/component: run
                apps.tanzu.vmware.com/auto-configure-actuators: "false"
                apps.tanzu.vmware.com/workload-type: web
                car
    ...
  supplyChainRef:
    kind: ClusterSupplyChain
    name: source-to-url
```

And with `tail`:

```
tanzu apps workload apply rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq-
sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:examp
le-rabbitmq-cluster-1" --type web --output yaml --yes --tail
+ rmq-sample-app-build-1-build-pod › prepare
rmq-sample-app-build-1-build-pod[prepare] Build reason(s): CONFIG
rmq-sample-app-build-1-build-pod[prepare] CONFIG:
rmq-sample-app-build-1-build-pod[prepare]       + env:
rmq-sample-app-build-1-build-pod[prepare]       + - name: BP_OCI_SOURCE
rmq-sample-app-build-1-build-pod[prepare]       +   value: main/103fde37882b5510e9b397
4e5fe209161b54f675
rmq-sample-app-build-1-build-pod[prepare]       resources: {}
rmq-sample-app-build-1-build-pod[prepare]       - source: {}
rmq-sample-app-build-1-build-pod[prepare]       + source:
rmq-sample-app-build-1-build-pod[prepare]       +   blob:
rmq-sample-app-build-1-build-pod[prepare]       +     url: http://fluxcd-source-contro
```

```
ller.flux-system.svc.cluster.local./gitrepository/default/rmq-sample-app/103fde37882b5
510e9b3974e5fe209161b54f675.tar.gz
rmq-sample-app-build-1-build-pod[prepare] Loading secret for "gcr.io" from secret "reg
istry-credentials" at location "/var/build-secrets/registry-credentials"
rmq-sample-app-build-1-build-pod[prepare] Loading secret for "registry.tanzu.vmware.co
m" from secret "registry-credentials" at location "/var/build-secrets/registry-credent
ials"
rmq-sample-app-build-1-build-pod[prepare] Loading cluster credential helpers
rmq-sample-app-build-1-build-pod[prepare] Downloading fluxcd-source-controller.flux-sy
stem.svc.cluster.local./gitrepository/default/rmq-sample-app/103fde37882b5510e9b3974e5
fe209161b54f675.tar.gz...
...
...
...
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2023-04-04T16:22:29Z"
  generation: 1
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-sample-app
  namespace: default
  resourceVersion: "184296857"
  uid: 7fa58a71-0b41-4975-b816-781b87d02cde
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
      url: https://github.com/jhvhs/rabbitmq-sample
status:
  conditions:
  ...
  - lastTransitionTime: "2023-04-04T16:26:03Z"
    message: ""
    reason: Ready
    status: "True"
    type: Ready
  observedGeneration: 1
  resources:
  - conditions:
    - lastTransitionTime: "2023-04-04T16:22:36Z"
      message: ""
      reason: ResourceSubmissionComplete
      status: "True"
      type: ResourceSubmitted
      ...
    name: source-provider
    outputs:
    - digest: sha256:1982253401b1be2236786e3da433216d36d289d0b158fbc9ca6477ac94879e60
      lastTransitionTime: "2023-04-04T16:22:36Z"
      name: url
      preview: |
        http://fluxcd-source-controller.flux-system.svc.cluster.local./gitrepository/d
efault/rmq-sample-app/103fde37882b5510e9b3974e5fe209161b54f675.tar.gz
    ...
    stampedRef:
      apiVersion: source.toolkit.fluxcd.io/v1beta1
```

```
      kind: GitRepository
      name: rmq-sample-app
      namespace: default
      resource: gitrepositories.source.toolkit.fluxcd.io
    templateRef:
      apiVersion: carto.run/v1alpha1
      kind: ClusterSourceTemplate
      name: source-template
  - conditions:
    ...
    - lastTransitionTime: "2023-04-04T16:25:45Z"
      message: ""
      reason: Ready
      status: "True"
      type: Ready
    inputs:
    - name: source-provider
    ...
  - conditions:
    ...
    - lastTransitionTime: "2023-04-04T16:25:52Z"
      message: ""
      reason: Ready
      status: "True"
      type: Ready
    inputs:
    - name: image-provider
    name: config-provider
    outputs:
    - digest: sha256:0549f3f3fe5ef817af62ae6357465e6df1a6c901e5a7abc17468ee3f3e16c1a1
      lastTransitionTime: "2023-04-04T16:25:52Z"
      name: config
      preview: |-
        metadata:
            annotations:
                boot.spring.io/version: 2.4.9
                conventions.carto.run/applied-conventions: |-
                    spring-boot-convention/auto-configure-actuators-check
                    ...
                developer.conventions/target-containers: workload
                services.conventions.carto.run/rabbitmq: amqp-client/5.10.0
            labels:
                app.kubernetes.io/component: run
                apps.tanzu.vmware.com/auto-configure-actuators: "false"
                apps.tanzu.vmware.com/workload-type: web
                car
    ...
  supplyChainRef:
    kind: ClusterSupplyChain
    name: source-to-url
```

Un-setting Git fields

There are various ways to update a workload. It can be by changing its fields through flags or create a `yaml` file with the changes and run `tanzu apps workload apply` command with the `--update-strategy` set as `replace` (check `--update-strategy` for a better usage explanation).

However, for fields deletion, there is an easier way supported for the `--git-*` flags in which, through setting them as empty string in the command, the `workload.spec.source.git` fields get removed.

For example, if there is a workload that specifies `--git-tag`, `--git-commit` and `--git-branch`, to remove any of these the only thing that needs to be done is use empty string right after setting them.

```
## Existing workload definition
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-sample-app
  namespace: default
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
        commit: dec60a68190a4a8ebd3644806962002983ded69e
        tag: v0.1.0
      url: https://github.com/jhvhs/rabbitmq-sample

## Update the workload to remove one of its git fields

tanzu apps workload apply rmq-sample-app --git-tag ""
  Update workload:
...
 17, 17   |    git:
 18, 18   |      ref:
 19, 19   |        branch: main
 20, 20   |        commit: dec60a68190a4a8ebd3644806962002983ded69e
 21     - |        tag: v0.1.0
 22, 21   |      url: https://github.com/jhvhs/rabbitmq-sample
  Really update the workload "rmq-sample-app"? [yN]: y
  Updated workload "rmq-sample-app"

To see logs:   "tanzu apps workload tail rmq-sample-app --timestamp --since 1h"
To get status: "tanzu apps workload get rmq-sample-app"

## Export the workload to see that `spec.source.git.ref.tag` is not part of the defini
tion

tanzu apps workload get rmq-sample-app --export

---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-sample-app
  namespace: default
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
```

```
        commit: dec60a68190a4a8ebd3644806962002983ded69e
      url: https://github.com/jhvhs/rabbitmq-sample
```

NOTE: If `--git-repo` is set to empty, then the whole git section is going to be removed from the workload definition.

```
tanzu apps workload apply rmq-sample-app --git-repo ""
  Update workload:
...
 12, 12   |    ref:
 13, 13   |      apiVersion: rabbitmq.com/v1beta1
 14, 14   |      kind: RabbitmqCluster
 15, 15   |      name: example-rabbitmq-cluster-1
 16     - |  source:
 17     - |    git:
 18     - |      ref:
 19     - |        branch: main
 20     - |        commit: dec60a68190a4a8ebd3644806962002983ded69e
 21     - |      url: https://github.com/jhvhs/rabbitmq-sample
  NOTICE: no source code or image has been specified for this workload.
  Really update the workload "rmq-sample-app"? [yN]: y
  Updated workload "rmq-sample-app"

To see logs:   "tanzu apps workload tail rmq-sample-app --timestamp --since 1h"
To get status: "tanzu apps workload get rmq-sample-app"

## Export the workload and check that the git source section does not exist

tanzu apps workload get rmq-sample-app --export
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-sample-app
  namespace: default
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
```

Remove color from output

Most of Tanzu Apps Plug-in commands have emojis and colored output with the intention to be more user-friendly.

However, sometimes color, emojis and other characters are not needed (e.g. scripting) or even well interpreted in certain terminals and the best way to suppress them is using the `--no-color` flag.

So, for example, in a workload that is created through local path, which usually shows emojis and a progress bar, these special characters would be avoid by using `--no-color`.

```
tanzu apps workload apply my-workload --local-path path/to/my/source -s my-registry.ex
t/my-project/my-workload --type web --no-color
The files and/or directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "path/to/my/source" to "my-registry.ext/my-project/my-workloa
d"...
Published source
```

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: my-workload
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    image: my-registry.ext/my-project/my-workload:latest@sha256:724bcd14c3a
84fc7a918cd8ee7a6a987de1699617a17c5af166e8c689a2becf7
? Do you want to create this workload? [yN]:
```

To avoid having color in general for Tanzu Apps Plug-in, the best way is setting the `NO_COLOR` environment variable. This will suppress color, emojis and progress bar for all the Plug-in related commands.

```
export NO_COLOR=true
```

Debug workloads

This topic tells you how to use the Tanzu Apps CLI to debug workloads.

Verify build logs

Check build logs

After a workload is created, tail the workload to view the build and runtime logs.

Check logs by running:

```
tanzu apps workload tail pet-clinic --since 10m --timestamp
```

Where:

- `pet-clinic` is the name you gave the workload.
- `--since` (optional) the amount of time to go back to begin streaming logs. The default is 1 second.
- `--timestamp` (optional) prints the timestamp with each log entry.

Get the workload status and details

After the workload build process is complete, create a Knative service to run the workload. You can view workload details at any time during the process. Some details, such as the workload URL, are only available after the workload is running.

To check the workload details, run:

```
tanzu apps workload get pet-clinic
```

Where:

`pet-clinic` is the name of the workload you want details about.

You can now see the running workload. When the workload is created, `tanzu apps workload get` includes the URL for the running workload. Some terminals allow you to `ctrl`+click the URL to view it. You can also copy and paste the URL into your web browser to see the workload.

Common workload errors

A workload can either be ready, on error or with an unknown status.

There are known errors that cause the workload to enter an error or unknown status. The most common are:

Local Path Development Error Cases

Message: Writing `registry/project/repo/workload:latest`: Writing image: Unexpected status code *401 Unauthorized* (HEAD responses have no body, use GET for details)

Cause: Apps plug-in cannot talk to the registry because the registry credentials are missing or invalid.

Resolution: Run `docker logout registry` and `docker login registry` commands and specify the valid credentials for the registry.

Message: Writing `registry/project/workload:latest`: Writing image: HEAD Unexpected status code *400 Bad Request* (HEAD responses have no body, use GET for details)

Cause: Certain registries like Harbor or GCR have a concept of `Project`. A 400 Bad request is sent when either the project does not exist, the user does not have access to it, or the path in the `–source-image` flag is missing either project or repository.

Resolution: Fix the path in the `–source-image` flag value to point to a valid repository path.

WorkloadLabelsMissing/SupplyChainNotFound

Message: No supply chain found where full selector is satisfied by `labels: map[app.kubernetes.io/part-of:spring-petclinic]`

Cause: The labels and attributes in the workload object did not fully satisfy any installed supply chain on the cluster.

Resolution: Use the `tanzu apps cluster-supply-chain list` (alias `csc`) and `tanzu apps csc get <supply-chain-name>` commands to see the workload selection criteria for the supply chain available on the cluster. Apply any missing labels to a workload by using `tanzu apps workload apply --label required-label-name=required-label-value`. For example:

```
tanzu apps workload apply workload-name –-type web
tanzu apps workload apply workload-name --label apps.tanzu.vmware.com/workload-type=we
b
```

MissingValueAtPath

Message: Waiting to read value `[.status.artifact.url]` from resource gitrepository.source.toolkit.fluxcd.io in namespace `[ns]`

Possible Cause: The Git `url/tag/branch/commit` parameters passed in the workload are not valid.

Resolution: Fix the invalid Git parameters by using `tanzu apps workload apply`

Possible Cause: The Git repository is not accessible from the cluster

Resolution: Configure your cluster networking or your Git repository networking so that they can communicate with each other.

Possible Cause: The namespace is missing the Git secret for communicating with the private repository

Resolution: For more information, see Git authentication

TemplateRejectedByAPIServer

Message: Unable to apply object `[ns/workload-name]` for resource `[source-provider]` in supply chain `[source-to-url]`: failed to get unstructured `[ns/workload-name]` from API server: imagerepositories.source.apps.tanzu.vmware.com “workload-name” is forbidden: User “system:serviceaccount:ns:default” cannot get resource “imagerepositories” in API group “source.apps.tanzu.vmware.com” in the namespace “ns”

Cause: This error happens when the service account in the workload object does not have permission to create objects that are stamped out by the supply chain.

Resolution: Set up the Set up developer namespaces to use your installed packages with the required service account and permissions.

Review supply chain steps

After you create a workload with the `tanzu apps workload create (or) apply`, command, you can run the `tanzu apps workload get` command to display the current condition of each supply chain.

For example:

```
...
  Supply Chain
   name:   source-to-url

   NAME                READY   HEALTHY   UPDATED   RESOURCE
   source-provider     True    True      71m       gitrepositories.source.toolkit.fluxc
d.io/spring-petclinic
   image-provider      True    True      70m       images.kpack.io/spring-petclinic
   config-provider     True    True      69m       podintents.conventions.carto.run/spr
ing-petclinic
   app-config          True    True      69m       configmaps/spring-petclinic
   service-bindings    True    True      69m       configmaps/spring-petclinic-with-cla
ims
   api-descriptors     True    True      69m       configmaps/spring-petclinic-with-api
-descriptors
   config-writer       True    True      69m       runnables.carto.run/spring-petclinic
-config-writer

  Delivery
   name:   delivery-basic

   NAME               READY   HEALTHY   UPDATED   RESOURCE
   source-provider    True    True      69m       imagerepositories.source.apps.tanzu.v
mware.com/spring-petclinic-delivery
   deployer           True    True      69m       apps.kappctrl.k14s.io/spring-petclini
c

  Messages
   No messages found.
...
```

The `Supply Chain` section displays the supply chain steps associated with the workload. If a step fails, the `READY` column value is `Unknown` or `False`, and the `HEALTHY` column value is `False`. If there is a resource in `Unknown` or `False` status, inspect it with:

```
kubectl describe RESOURCE-NAME
```

Where `RESOURCE-NAME` refers to the name of the stamped out resource, displayed in `RESOURCE` column.

For example, if `tanzu apps workload get` command returns this resource:

```
NAME                READY   HEALTHY   UPDATED   RESOURCE
source-provider     False   False     3h12m     gitrepositories.source.toolkit.fluxcd.i
o/spring-petclinic
```

Check this resource with:

```
kubectl describe gitrepositories.source.toolkit.fluxcd.io/spring-petclinic
```

The `Messages` section might give a hint as to what went wrong in the process. For example, a message similar to the following is shown:

```
  Messages
   Workload [HealthyConditionRule]:   failed to checkout and determine revision: faile
d to resolve commit object for '425ae9a2a2f84d195a9f3862668e8b2abf81418a': object not
found
```

This might mean that the commit does not belong to the specified branch or does not exist in the repository.

Additional Troubleshooting References

For more workload troubleshooting tips, see Troubleshoot using Tanzu Application Platform page.

Tanzu Apps CLI commands

The following topics describe the Tanzu CLI Apps plug-in commands.

- tanzu apps clustersupplychain sub-commands and details.
- `tanzu apps workload` sub-commands and flags usage for each:
  - tanzu apps workload get
  - tanzu apps workload list
  - tanzu apps workload tail
  - tanzu apps workload delete
  - tanzu apps workload apply

Tanzu Apps CLI commands

The following topics describe the Tanzu CLI Apps plug-in commands.

- tanzu apps clustersupplychain sub-commands and details.
- `tanzu apps workload` sub-commands and flags usage for each:
  - tanzu apps workload get
  - tanzu apps workload list
  - tanzu apps workload tail
  - tanzu apps workload delete
  - tanzu apps workload apply

tanzu apps cluster-supply-chain

This topic tells you about the Tanzu Apps CLI `cluster-supply-chain` command.

Tanzu apps cluster supply chain list

The `tanzu apps clustersupplychain list` command lists the available supply chains installed in the cluster (supported clustersupplychain alias is `csc`).

Run the following command to view more detailed information about the selectors and conditions that must be met for a workload to be selected by a certain supply chain:

```
tanzu apps clustersupplychain get SUPPLYCHAIN-NAME`.
```

Default view

The default view displays the name of the supply chain, whether it is ready or not, and its age.

For example:

```
tanzu apps clustersupplychain list
NAME                 READY   AGE
basic-image-to-url   Ready   11d
source-to-url        Ready   11d

To view details: "tanzu apps cluster-supply-chain get <name>"
```

Tanzu apps cluster supply chain get

The `tanzu apps clustersupplychain get` command gets detailed information of the cluster supply chain.

Default view

The default view displays the status of the supply chain, and the selectors that a workload must match so it is taken by that supply chain.

For example:

```
tanzu apps cluster-supply-chain get source-to-url
---
# source-to-url: Ready
---
Supply Chain Selectors
   TYPE          KEY                                    OPERATOR   VALUE
   expressions   apps.tanzu.vmware.com/workload-type   In         web
   expressions   apps.tanzu.vmware.com/workload-type   In         server
   expressions   apps.tanzu.vmware.com/workload-type   In         worker
```

This output indicates the attributes a workload needs to be selected by the `source-to-url` supply chain on the target cluster. For example:

- The workload must have the `--type` flag value of `web`, `server`, or `worker`.
- Or, if expressed through `workload.yaml`, the `Workload.metadata.labels` label `apps.tanzu.vmware.com/workload-type` must exist and have a value of `web`, `server` , or `worker`.

Another example is the `testing/scanning` pipeline, which has the `tekton` steps for testing and the scanning steps.

```
---
# source-test-scan-to-url: Ready
---
Supply Chain Selectors
   TYPE          KEY                                    OPERATOR   VALUE
   labels        apps.tanzu.vmware.com/has-tests                   true
   expressions   apps.tanzu.vmware.com/workload-type   In         web
   expressions   apps.tanzu.vmware.com/workload-type   In         server
   expressions   apps.tanzu.vmware.com/workload-type   In         worker
```

In this case, the workload must have both labels `apps.tanzu.vmware.com/has-tests: true` and `apps.tanzu.vmware.com/workload-type` set up as `web`, `server`, or `worker` to be selected for the supply chain.

tanzu apps workload apply

This topic tells you about the Tanzu Apps CLI `tanzu apps workload apply` command.

Use the `tanzu apps workload apply` command to create and update workloads that are deployed in a cluster through a supply chain.

The `tanzu apps workload apply` and `tanzu apps workload create` commands have the same behavior and flags with the following exceptions:

- The `tanzu apps workload create` command fails if a workload with the same name preexists on the target cluster.
- the `update-strategy` flag is only applicable to the `tanzu apps workload apply` command. The `update-strategy` flag is not applicable to the `tanzu apps workload create` command.

Default view

In the output of the `tanzu apps workload apply` command, the specification for the workload is shown in YAML file format.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.5.0 --
type web
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    git:
     12 + |      ref:
     13 + |        tag: tap-1.5.0
     14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     15 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
  Created workload "tanzu-java-web-app"
```

```
To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"
```

In the first section, the definition of workload is displayed. It's followed by a dialog box asking `whether the workload should be created or updated`. In the last section, if a workload is created or updated, some hints are displayed about the next steps.

Workload Apply flags

`--annotation`

Sets the annotations to be applied to the workload. To specify more than one annotation set the flag multiple times. These annotations are passed as parameters to be processed in the supply chain.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.5.0 --
type web --annotation tag=tap-1.5.0 --annotation name="Tanzu Java Web"
  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    apps.tanzu.vmware.com/workload-type: web
    7 + |  name: tanzu-java-web-app
    8 + |  namespace: default
    9 + |spec:
   10 + |  params:
   11 + |  - name: annotations
   12 + |    value:
   13 + |      name: Tanzu Java Web
   14 + |      tag: tap-1.5.0
   15 + |  source:
   16 + |    git:
   17 + |      ref:
   18 + |        tag: tap-1.5.0
   19 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
   20 + |    subPath: tanzu-java-web-app
```

To delete an annotation, use `-` after its name.

Example

```
tanzu apps workload apply tanzu-java-web-app --annotation tag-
  Update workload:
...
10, 10   |  params:
11, 11   |  - name: annotations
12, 12   |    value:
13, 13   |      name: Tanzu Java Web
14     - |      tag: tap-1.5.0
15, 14   |  source:
16, 15   |    git:
17, 16   |      ref:
18, 17   |        tag: tap-1.5.0
...
  Really update the workload "tanzu-java-web-app"? [yN]:
```

`--app` / `-a`

This is the application the workload is part of. This is part of the workload metadata section.

Example

```
tanzu apps workload apply tanzu-app --git-repo https://github.com/vmware-tanzu/applica
tion-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.5.0 --type web
--app tanzu-java-web-app
  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
    7 + |    apps.tanzu.vmware.com/workload-type: web
    8 + |  name: tanzu-app
    9 + |  namespace: default
   10 + |spec:
   11 + |  source:
   12 + |    git:
   13 + |      ref:
   14 + |        tag: tap-1.5.0
   15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
   16 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
  Created workload "tanzu-app"

To see logs:   "tanzu apps workload tail tanzu-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-app"
```

`--build-env`

Sets environment variables to use in the build phase by the build resources in the supply chain.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.5.0 --
type web --build-env JAVA_VERSION=1.8
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  build:
     11 + |    env:
     12 + |    - name: JAVA_VERSION
     13 + |      value: "1.8"
     14 + |  source:
     15 + |    git:
     16 + |      ref:
     17 + |        tag: tap-1.5.0
     18 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     19 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

To delete a build environment variable, use `-` after its name.

Example

```
tanzu apps workload apply tanzu-java-web-app --build-env JAVA_VERSION-
  Update workload:
...
   6,  6   |    apps.tanzu.vmware.com/workload-type: web
   7,  7   |  name: tanzu-java-web-app
   8,  8   |  namespace: default
   9,  9   |spec:
  10     - |  build:
  11     - |    env:
  12     - |    - name: JAVA_VERSION
  13     - |      value: "1.8"
  14, 10   |  source:
  15, 11   |    git:
  16, 12   |      ref:
  17, 13   |        tag: tap-1.5.0
...
  Really update the workload "tanzu-java-web-app"? [yN]:
```

--debug

Sets the parameter variable debug to true in the workload.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --debug
  Create workload:
     1 + |---
     2 + |apiVersion: carto.run/v1alpha1
     3 + |kind: Workload
     4 + |metadata:
     5 + |  labels:
     6 + |    apps.tanzu.vmware.com/workload-type: web
     7 + |  name: tanzu-java-web-app
     8 + |  namespace: default
     9 + |spec:
    10 + |  params:
    11 + |  - name: debug
    12 + |    value: "true"
    13 + |  source:
    14 + |    git:
    15 + |      ref:
    16 + |        branch: main
    17 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
    18 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

--dry-run

Prepares all the steps to submit the workload to the cluster and stops before sending it, showing an output of the final structure of the workload.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.5.0 --
type web --build-env JAVA_VERSION=1.8 --param-yaml server=$'port: 8080\nmanagement-por
t: 8181' --dry-run
---
apiVersion: carto.run/v1alpha1
```

```
kind: Workload
metadata:
  creationTimestamp: null
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: default
spec:
  build:
    env:
    - name: JAVA_VERSION
      value: "1.8"
  params:
  - name: server
    value:
      management-port: 8181
      port: 8080
  source:
    git:
      ref:
        tag: tap-1.5.0
      url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: tanzu-java-web-app
status:
  supplyChainRef: {}
```

`--env` / `-e`

Sets the environment variables to the workload so the supply chain resources can use it to deploy the workload application.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.5.0 --
type web --env NAME="Tanzu Java App"
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  env:
     11 + |  - name: NAME
     12 + |    value: Tanzu Java App
     13 + |  source:
     14 + |    git:
     15 + |      ref:
     16 + |        tag: tap-1.5.0
     17 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     18 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

To unset an environment variable, use `-` after its name.

```
tanzu apps workload apply tanzu-java-web-app --env NAME-
  Update workload:
...
   6,  6   |    apps.tanzu.vmware.com/workload-type: web
   7,  7   |  name: tanzu-java-web-app
```

```
   8,  8   |  namespace: default
   9,  9   |spec:
  10      - |  env:
  11      - |  - name: NAME
  12      - |    value: Tanzu Java App
  13, 10   |  source:
  14, 11   |    git:
  15, 12   |      ref:
  16, 13   |        tag: tap-1.5.0
...
  Really update the workload "tanzu-java-web-app"? [yN]:
```

`--file, -f`

Sets the workload specification file to create the workload. This comes from any other workload specification passed by flags to the command set or overrides what is in the file. Another way to use this flag is by using `-` in the command to receive workload definition through stdin. See Working with YAML Files for an example.

Example

```
tanzu apps workload apply tanzu-java-web-app -f java-app-workload.yaml --param-yaml se
rver=$'port: 9090\nmanagement-port: 9190'
  Create workload:
       1 + |---
       2 + |apiVersion: carto.run/v1alpha1
       3 + |kind: Workload
       4 + |metadata:
       5 + |  labels:
       6 + |    apps.tanzu.vmware.com/workload-type: web
       7 + |  name: tanzu-java-web-app
       8 + |  namespace: default
       9 + |spec:
      10 + |  build:
      11 + |    env:
      12 + |    - name: JAVA_VERSION
      13 + |      value: "1.8"
      14 + |  params:
      15 + |  - name: server
      16 + |    value:
      17 + |      management-port: 9190
      18 + |      port: 9090
      19 + |  source:
      20 + |    git:
      21 + |      ref:
      22 + |        tag: tap-1.5.0
      23 + |      url: url: https://github.com/vmware-tanzu/application-accelerator-sa
mples
      24 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

`--git-repo`

The Git repository from which the workload is created. With this, either `--git-tag`, `--git-commit`, `--git-branch` or the three of them can be specified. When setting this flag to empty string, the whole `spec.source.git` section is removed from workload definition.

For Git source, if all the flags are specified (`--git-tag`, `--git-commit`, `--git-branch`) the revision to which the workload will checkout depends on the source controller.

`--git-branch`

The branch in a Git repository from where the workload is created. Commit and tag can also be specified alongside this flag. It can be unset by defining it as empty string when applying a workload (`--git-branch ""`).

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web
  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    apps.tanzu.vmware.com/workload-type: web
    7 + |  name: tanzu-java-web-app
    8 + |  namespace: default
    9 + |spec:
   10 + |  source:
   11 + |    git:
   12 + |      ref:
   13 + |        branch: main
   14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
   15 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

`--git-tag`

The tag in a Git repository from which the workload is created. Can be unset by defining it as empty string when applying a workload (`--git-tag ""`).

`--git-commit`

Commit in Git repository from where the workload is resolved. Either `--git-branch` or `--git-tag` can be specified with it too. It can be unset by defining it as empty string when applying a workload (`--git-commit ""`).

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-commit 1c4cf82e4
99f7e46da182922d4097908d4817320 --type web
  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    apps.tanzu.vmware.com/workload-type: web
    7 + |  name: tanzu-java-web-app
    8 + |  namespace: default
    9 + |spec:
   10 + |  source:
   11 + |    git:
   12 + |      ref:
   13 + |        commit: 1c4cf82e499f7e46da182922d4097908d4817320
   14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
   15 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

`--image` / `-i`

Sets the OSI image to be used as the workload application source instead of a Git repository

Example

```
tanzu apps workload apply tanzu-java-web-app --image private.repo.domain.com/tanzu-jav
a-web-app --type web
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  build:
     11 + |    env:
     12 + |    - name: JAVA_VERSION
     13 + |      value: "1.8"
     14 + |  params:
     15 + |  - name: server
     16 + |    value:
     17 + |      management-port: 9190
     18 + |      port: 9090
     19 + |  source:
     20 + |    git:
     21 + |      ref:
     22 + |        tag: tap-1.5.0
     23 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     24 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

`--label` / `-l`

Sets the label to be applied to the workload. To specify more than one label, set the flag multiple times.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --label stage=production
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    stage: production
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  source:
     12 + |    git:
     13 + |      ref:
     14 + |        branch: main
     15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     16 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

To unset labels, use `-` after their name.

Example

```
tanzu apps workload apply tanzu-java-web-app --label stage-
  Update workload:
...
   3,  3   |kind: Workload
   4,  4   |metadata:
   5,  5   |  labels:
   6,  6   |    apps.tanzu.vmware.com/workload-type: web
   7     - |    stage: production
   8,  7   |  name: tanzu-java-web-app
   9,  8   |  namespace: default
  10,  9   |spec:
  11, 10   |  source:
...
  Really update the workload "tanzu-java-web-app"? [yN]:
```

`--limit-cpu`

The maximum CPU the workload pods are allowed to use.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --limit-cpu .2
  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    apps.tanzu.vmware.com/workload-type: web
    7 + |  name: tanzu-java-web-app
    8 + |  namespace: default
    9 + |spec:
   10 + |  resources:
   11 + |    limits:
   12 + |      cpu: 200m
   13 + |  source:
   14 + |    git:
   15 + |      ref:
   16 + |        branch: main
   17 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
   18 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

`--limit-memory`

The maximum memory the workload pods are allowed to use.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --limit-memory 200Mi
  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    apps.tanzu.vmware.com/workload-type: web
    7 + |  name: tanzu-java-web-app
    8 + |  namespace: default
    9 + |spec:
```

```
  10 + |  resources:
  11 + |    limits:
  12 + |      memory: 200Mi
  13 + |  source:
  14 + |    git:
  15 + |      ref:
  16 + |        branch: main
  17 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
  18 + |    subPath: tanzu-java-web-app
Do you want to create this workload? [yN]:
```

`--live-update`

Enable this to deploy the workload once, save changes to the code, and see those changes reflected in the workload running on the cluster.

Example

An example with a Spring Boot application:

1. Clone the repository by running:

   ```
   git clone https://github.com/vmware-tanzu/application-accelerator-samples
   ```

2. Change into the `tanzu-java-web-app` directory.
3. In `Tiltfile`, first change the `SOURCE_IMAGE` variable to use your registry and project.
4. At the very end of the file add:

   ```
   allow_k8s_contexts('your-cluster-name')
   ```

5. Inside the directory, run:

   ```
   tanzu apps workload apply tanzu-java-web-app --live-update --local-path . -s
   gcr.io/my-project/tanzu-java-web-app-live-update -y
   ```

   Expected output:

   ```
   The files and directories listed in the .tanzuignore file are being excluded fr
   om the uploaded source code.
   Publishing source in "." to "gcr.io/my-project/tanzu-java-web-app-live-updat
   e"...
     Published source

     Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  name: tanzu-java-web-app
      6 + |  namespace: default
      7 + |spec:
      8 + |  params:
      9 + |  - name: live-update
     10 + |    value: "true"
     11 + |  source:
     12 + |    image: gcr.io/my-project/tanzu-java-web-app-live-update:latest@sha2
   56:3c9fd738492a23ac532a709301fcf0c9aa2a8761b2b9347bdbab52ce9404264b

     Created workload "tanzu-java-web-app"

   To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since
   1h"
   ```

```
To get status: "tanzu apps workload get tanzu-java-web-app"
```

6. Run Tilt to deploy the workload.

```
tilt up

Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

(space) to open the browser
(s) to stream logs (--stream=true)
(t) to open legacy terminal mode (--legacy=true)
(ctrl-c) to exit
Tilt started on http://localhost:10350/
v0.23.6, built 2022-01-14

Initial Build • (Tiltfile)
Loading Tiltfile at: /path/to/repo/tanzu-java-web-app/Tiltfile
Successfully loaded Tiltfile (1.500809ms)
tanzu-java-w… |
tanzu-java-w… | Initial Build • tanzu-java-web-app
tanzu-java-w… | WARNING: Live Update failed with unexpected error:
tanzu-java-w… |   Cannot extract live updates on this build graph structure
tanzu-java-w… | Falling back to a full image build + deploy
tanzu-java-w… | STEP 1/1 — Deploying
tanzu-java-w… |       Objects applied to cluster:
tanzu-java-w… |         → tanzu-java-web-app:workload
tanzu-java-w… |
tanzu-java-w… |       Step 1 - 8.87s (Deploying)
tanzu-java-w… |       DONE IN: 8.87s
tanzu-java-w… |
tanzu-java-w… |
tanzu-java-w… | Tracking new pod rollout (tanzu-java-web-app-build-1-build-po
d):
tanzu-java-w… |       ┊ Scheduled       - (…) Pending
tanzu-java-w… |       ┊ Initialized     - (…) Pending
tanzu-java-w… |       ┊ Ready           - (…) Pending
...
```

`--local-path`

Sets the path to a source in the local machine from where the workload creates an image to use as an application source. The local path can be a directory, a JAR, a ZIP, or a WAR file. Java/Spring Boot compiled binaries are also supported. This flag must be used with `--source-image` flag.

If Java/Spring compiled binary is passed instead of source code, the command takes less time to apply the workload since the build pack skips the compiling steps and start uploading the image.

When working with local source code, you can exclude files from the source code to be uploaded within the image by creating a file `.tanzuignore` at the root of the source code. The `.tanzuignore` file contains a list of file paths to exclude from the image including the file itself. The directories must not end with the system path separator (`/` or `\`). If the file contains directories that are not in the source code, they are ignored. Lines starting with a `#` hashtag are also ignored.

`--maven-artifact`

This artifact is an output of a Maven project build. This flag must be used with `--maven-version` and `--maven-group`.

Example

```
tanzu apps workload apply petc-mvn --maven-artifact petc --maven-version 2.6.1 --maven
-group demo.com
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  name: petc-mvn
      6 + |  namespace: default
      7 + |spec:
      8 + |  params:
      9 + |  - name: maven
     10 + |    value:
     11 + |      artifactId: petc
     12 + |      groupId: demo.com
     13 + |      version: 2.6.1
  Do you want to create this workload? [yN]:
```

`--maven-group`

This group identifies the project across all other Maven projects.

`--maven-type`

This specifies the type of artifact that the Maven project produces. This flag is optional and is set by default as `jar` by the supply chain.

`--maven-version`

Definition of the current version of the Maven project.

`--source-image`, `-s`

Registry path where the local source code is uploaded as an image.

Example

```
tanzu apps workload apply spring-pet-clinic --local-path /home/user/workspace/spring-p
et-clinic --source-image gcr.io/spring-community/spring-pet-clinic --type web
  Publish source in "/home/user/workspace/spring-pet-clinic" to "gcr.io/spring-communi
ty/spring-pet-clinic"? It might be visible to others who can pull images from that rep
ository Yes
The files and/or directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "/home/user/workspace/spring-pet-clinic" to "gcr.io/spring-commun
ity/spring-pet-clinic"...
  Published source

  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: spring-pet-clinic
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    image:gcr.io/spring-community/spring-pet-clinic:latest@sha256:5feb0d9da
f3f639755d8683ca7b647027cfddc7012e80c61dcdac27f0d7856a7
  Do you want to create this workload? [yN]:
```

`--namespace`, `-n`

Specifies the namespace in which the workload is created or updated in.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --namespace my-namespace
  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    apps.tanzu.vmware.com/workload-type: web
    7 + |  name: tanzu-java-web-app
    8 + |  namespace: my-namespace
    9 + |spec:
   10 + |  source:
   11 + |    git:
   12 + |      ref:
   13 + |        branch: main
   14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
  15 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

`--output`, `-o`

This flag can be used to retrieve a workload right after it's applied in the specified format (`yaml`, `yml`, `json`). If used with `--yes` flag, all prompts are skipped and it only returns the workload definition. It can also be used with `--wait` or `--tail` flags to return the workload with its status.

Example

```
tanzu apps workload apply rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq-
sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:examp
le-rabbitmq-cluster-1" --type web --output yaml
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: rmq-sample-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  serviceClaims:
     11 + |  - name: rmq
     12 + |    ref:
     13 + |      apiVersion: rabbitmq.com/v1beta1
     14 + |      kind: RabbitmqCluster
     15 + |      name: example-rabbitmq-cluster-1
     16 + |  source:
     17 + |    git:
     18 + |      ref:
     19 + |        branch: main
     20 + |      url: https://github.com/jhvhs/rabbitmq-sample
  Do you want to create this workload? [yN]: y
  Created workload "rmq-sample-app"

To see logs:   "tanzu apps workload tail rmq-sample-app --timestamp --since 1h"
To get status: "tanzu apps workload get rmq-sample-app"
```

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  creationTimestamp: "2023-04-04T15:18:13Z"
  generation: 1
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: rmq-sample-app
  namespace: default
  resourceVersion: "184169566"
  uid: 6588d398-b803-47e3-b31a-23d9a1a633a9
spec:
  serviceClaims:
  - name: rmq
    ref:
      apiVersion: rabbitmq.com/v1beta1
      kind: RabbitmqCluster
      name: example-rabbitmq-cluster-1
  source:
    git:
      ref:
        branch: main
      url: https://github.com/jhvhs/rabbitmq-sample
status:
  supplyChainRef: {}
```

`--param` / `-p`

Additional parameters to be sent to the supply chain, the value is sent as a string. For complex YAML and JSON objects use `--param-yaml`.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --param port=9090 --param management-port=9190
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  params:
     11 + |  - name: port
     12 + |    value: "9090"
     13 + |  - name: management-port
     14 + |    value: "9190"
     15 + |  source:
     16 + |    git:
     17 + |      ref:
     18 + |        branch: main
     19 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     20 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

To unset parameters, use `-` after their name.

Example

```
tanzu apps workload apply tanzu-java-web-app --param port-
  Update workload:
...
   7,  7   |  name: tanzu-java-web-app
   8,  8   |  namespace: default
   9,  9   |spec:
  10, 10   |  params:
  11     - |  - name: port
  12     - |    value: "9090"
  13, 11   |  - name: management-port
  14, 12   |    value: "9190"
  15, 13   |  source:
  16, 14   |    git:
...
  Really update the workload "tanzu-java-web-app"? [yN]:
```

--param-yaml

Additional parameters to be sent to the supply chain, the value is sent as a complex object.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --param-yaml server=$'port: 9090\nmanagement-port: 9190'
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  params:
     11 + |  - name: server
     12 + |    value:
     13 + |      management-port: 9190
     14 + |      port: 9090
     15 + |  source:
     16 + |    git:
     17 + |      ref:
     18 + |        branch: main
     19 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     20 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

To unset parameters, use `-` after their name.

Example

```
tanzu apps workload apply tanzu-java-web-app --param-yaml server-
  Update workload:
...
   6,  6   |    apps.tanzu.vmware.com/workload-type: web
   7,  7   |  name: tanzu-java-web-app
   8,  8   |  namespace: default
   9,  9   |spec:
  10     - |  params:
  11     - |  - name: server
  12     - |    value:
  13     - |      management-port: 9190
  14     - |      port: 9090
```

```
  15, 10   |  source:
  16, 11   |    git:
  17, 12   |      ref:
  18, 13   |        branch: main
...
  Really update the workload "tanzu-java-web-app"? [yN]:
```

`--registry-ca-cert`

Refers to the path of the self-signed certificate needed for the custom/private registry. This is also populated with a default value through environment variables. If the environment variable `TANZU_APPS_REGISTRY_CA_CERT` is set, it's not necessary to use it in the command.

See Custom registry credentials for the supported environment variables.

Example

```
tanzu apps workload apply my-workload --local-path . -s registry.url.nip.io/my-packag
e/my-image --type web --registry-ca-cert path/to/cacert/mycert.nip.io.crt --registry-u
sername my-username --registry-password my-password
  Publish source in "." to "registry.url.nip.io/my-package/my-image"? It might be visi
ble to others who can pull images from that repository Yes
Publishing source in "." to "registry.url.nip.io/my-package/my-image"...
  Published source

  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: my-workload
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    image: registry.url.nip.io/my-package/my-image:latest@sha256:caeb7e3a0e
3ae0659f74d01095b6fdfe0d3c4a12856a15ac67ad6cd3b9e43648
  Do you want to create this workload? [yN]:
```

`--registry-password`

If credentials are needed, the user name and password values are set through the `--registry-password` flag. The value of this flag can also be specified through `TANZU_APPS_REGISTRY_PASSWORD`.

`--registry-token`

Used for token authentication in the private registry. This flag is set as `TANZU_APPS_REGISTRY_TOKEN` environment variable.

`--registry-username`

Often used with `--registry-password` to set private registry credentials. Can be provided using `TANZU_APPS_REGISTRY_USERNAME` environment variable to avoid setting it every time in the command.

`--request-cpu`

Refers to the minimum CPU the workload pods request to use.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --request-cpu .3
  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    apps.tanzu.vmware.com/workload-type: web
    7 + |  name: tanzu-java-web-app
    8 + |  namespace: default
    9 + |spec:
   10 + |  resources:
   11 + |    requests:
   12 + |      cpu: 300m
   13 + |  source:
   14 + |    git:
   15 + |      ref:
   16 + |        branch: main
   17 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
   18 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

--request-memory

Refers to the minimum memory the workload pods are requesting to use.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --request-memory 300Mi
  Create workload:
     1 + |---
     2 + |apiVersion: carto.run/v1alpha1
     3 + |kind: Workload
     4 + |metadata:
     5 + |  labels:
     6 + |    apps.tanzu.vmware.com/workload-type: web
     7 + |  name: tanzu-java-web-app
     8 + |  namespace: default
     9 + |spec:
    10 + |  resources:
    11 + |    requests:
    12 + |      memory: 300Mi
    13 + |  source:
    14 + |    git:
    15 + |      ref:
    16 + |        branch: main
    17 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
    18 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

--service-account

Refers to the service account to associate with the workload. A service account provides an identity for a workload object.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
```

```
pe web --service-account petc-serviceaccount
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  serviceAccountName: petc-serviceaccount
     11 + |  source:
     12 + |    git:
     13 + |      ref:
     14 + |        branch: main
     15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     16 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]:
```

To unset a service account, pass empty string.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --service-account ""
  Update workload:
...
  6,  6   |    apps.tanzu.vmware.com/workload-type: web
  7,  7   |  name: tanzu-java-web-app
  8,  8   |  namespace: default
  9,  9   |spec:
 10      - |  serviceAccountName: petc-serviceaccount
 11, 10   |  source:
 12, 11   |    git:
 13, 12   |      ref:
 14, 13   |        branch: main
...
  Really update the workload "tanzu-java-web-app"? [yN]:
```

--service-ref

Binds a service to a workload to provide the information from a service resource to an application.

For more information, see Tanzu Application Platform documentation.

Example

```
tanzu apps workload apply rmq-sample-app --git-repo https://github.com/jhvhs/rabbitmq-
sample --git-branch main --service-ref "rmq=rabbitmq.com/v1beta1:RabbitmqCluster:examp
le-rabbitmq-cluster-1"
  Create workload:
     1 + |---
     2 + |apiVersion: carto.run/v1alpha1
     3 + |kind: Workload
     4 + |metadata:
     5 + |  name: rmq-sample-app
     6 + |  namespace: default
     7 + |spec:
     8 + |  serviceClaims:
     9 + |  - name: rmq
    10 + |    ref:
    11 + |      apiVersion: rabbitmq.com/v1beta1
    12 + |      kind: RabbitmqCluster
```

```
    13 + |      name: example-rabbitmq-cluster-1
    14 + |  source:
    15 + |    git:
    16 + |      ref:
    17 + |        branch: main
    18 + |      url: https://github.com/jhvhs/rabbitmq-sample
  Do you want to create this workload? [yN]:
```

To delete service binding, use the service name followed by `-`.

Example

```
tanzu apps workload apply rmq-sample-app --service-ref rmq-
  Update workload:
...
   4,  4   |metadata:
   5,  5   |  name: rmq-sample-app
   6,  6   |  namespace: default
   7,  7   |spec:
   8      - |  serviceClaims:
   9      - |  - name: rmq
  10      - |    ref:
  11      - |      apiVersion: rabbitmq.com/v1beta1
  12      - |      kind: RabbitmqCluster
  13      - |      name: example-rabbitmq-cluster-1
  14,  8   |  source:
  15,  9   |    git:
  16, 10   |      ref:
  17, 11   |        branch: main
...
  Really update the workload "rmq-sample-app"? [yN]:
```

`--sub-path`

Defines which path is used as the root path to create and update the workload.

Example

- Git repository

  ```
  tanzu apps workload apply subpathtester --git-repo https://github.com/path-to-r
  epo/my-repo --git-branch main --type web --sub-path my-subpath
    Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: subpathtester
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    git:
     12 + |      ref:
     13 + |        branch: main
     14 + |      url: https://github.com/path-to-repo/my-repo
     15 + |    subPath: my-subpath
    Do you want to create this workload? [yN]:
  ```

- Local path
  - In the directory of the project you want to create the workload from

```
tanzu apps workload apply my-workload --local-path . -s gcr.io/my-registr
y/my-workload-image --sub-path subpath_folder
  Publish source in "." to "gcr.io/my-registry/my-workload-image"? It mig
ht be visible to others who can pull images from that repository Yes
Publishing source in "." to "gcr.io/my-registry/my-workload-image"...
  Published source

  Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  name: myworkload
    6 + |  namespace: default
    7 + |spec:
    8 + |  source:
    9 + |    image: gcr.io/my-registry/my-workload-image:latest@sha256:f2
8c5fedd0e902800e6df9605ce5e20a8e835df9e87b1a0aa256666ea179fc3f
   10 + |    subPath: subpath_folder
  Do you want to create this workload? [yN]:
```

`--tail`

Prints the logs of the workload creation in every step.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --tail
  Create workload:
     1 + |---
     2 + |apiVersion: carto.run/v1alpha1
     3 + |kind: Workload
     4 + |metadata:
     5 + |  labels:
     6 + |    apps.tanzu.vmware.com/workload-type: web
     7 + |  name: tanzu-java-web-app
     8 + |  namespace: default
     9 + |spec:
    10 + |  source:
    11 + |    git:
    12 + |      ref:
    13 + |        branch: main
    14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
    15 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]: y
  Created workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

Waiting for workload "tanzu-java-web-app" to become ready...
+ tanzu-java-web-app-build-1-build-pod › prepare
tanzu-java-web-app-build-1-build-pod[prepare] Build reason(s): CONFIG
tanzu-java-web-app-build-1-build-pod[prepare] CONFIG:
tanzu-java-web-app-build-1-build-pod[prepare]   + env:
tanzu-java-web-app-build-1-build-pod[prepare]   + - name: BP_OCI_SOURCE
tanzu-java-web-app-build-1-build-pod[prepare]   +   value: main/d381fb658cb435a04e2271
ca85bd3e8627a5e7e4
tanzu-java-web-app-build-1-build-pod[prepare]   resources: {}
tanzu-java-web-app-build-1-build-pod[prepare]   - source: {}
tanzu-java-web-app-build-1-build-pod[prepare]   + source:
tanzu-java-web-app-build-1-build-pod[prepare]   +   blob:
```

```
tanzu-java-web-app-build-1-build-pod[prepare]   +     url: http://source-controller.fl
ux-system.svc.cluster.local./gitrepository/default/tanzu-java-web-app/1c4cf82e499f7e46
da182922d4097908d4817320.tar.gz
...
...
...
```

--tail-timestamp

Prints the logs of the workload creation in every step adding the time in which the log is occurring.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web --tail-timestamp
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    git:
     12 + |      ref:
     13 + |        branch: main
     14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     15 + |    subPath: tanzu-java-web-app
  Do you want to create this workload? [yN]: y
  Created workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

Waiting for workload "tanzu-java-web-app" to become ready...
+ tanzu-java-web-app-build-1-build-pod › prepare
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.348418803-05:00 Buil
d reason(s): CONFIG
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364719405-05:00 CONF
IG:
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364761781-05:00   +
env:
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364771861-05:00   +
- name: BP_OCI_SOURCE
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364781718-05:00   +
value: main/d381fb658cb435a04e2271ca85bd3e8627a5e7e4
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364788374-05:00   re
sources: {}
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.364795451-05:00   -
source: {}
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.365344965-05:00   +
source:
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.365364101-05:00   +
blob:
tanzu-java-web-app-build-1-build-pod[prepare] 2022-06-15T11:28:01.365372427-05:00   +
url: http://source-controller.flux-system.svc.cluster.local./gitrepository/default/tan
zu-java-web-app/1c4cf82e499f7e46da182922d4097908d4817320.tar.gz
...
...
...
```

`--type` / `-t`

Sets the type of workload by adding the label `apps.tanzu.vmware.com/workload-type`, which is used as a matcher by supply chains. Use the `TANZU_APPS_TYPE` environment variable to have a default value for this flag.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-branch main --ty
pe web
  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: tanzu-java-web-app
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    git:
     12 + |      ref:
     13 + |        branch: main
     14 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     15 + |    subPath: tanzu-java-web-app
```

`--update-strategy`

Specifies whether the update from file should replace or merge the current workload. The default is merge.

Note

This flag is only applicable to the `tanzu apps workload apply` command. It is not applicable to the `tanzu apps workload create` command.

Example

For example, there is a workload created from a file, which has in its `spec` the following:

```
...
spec:
  resources:
    requests:
      memory: 1Gi
    limits:           # delete this line
      memory: 1Gi     # delete this line
      cpu: 500m       # delete this line
...
```

If the workload file is changed as specified in the comments, there are two ways to update the workload running in the cluster.

One, with `merge` update strategy.

```
tanzu apps workload apply -f ./spring-petclinic.yaml # defaulting to merge

  WARNING: Configuration file update strategy is changing. By default, provided config
uration files will replace rather than merge existing configuration. The change will t
```

```
ake place in the January 2024 TAP release (use "--update-strategy" to control strategy
explicitly).

Workload is unchanged, skipping update
```

The other, with `replace` update strategy, which completely overwrites the workload in the cluster according to the new specifications in the file.

```
tanzu apps workload apply -f ./spring-petclinic.yaml --update-strategy replace

  WARNING: Configuration file update strategy is changing. By default, provided config
uration files will replace rather than merge existing configuration. The change will t
ake place in the January 2024 TAP release (use "--update-strategy" to control strategy
explicitly).

  Update workload:
...
  8,  8   |  name: spring-petclinic
  9,  9   |  namespace: default
 10, 10   |spec:
 11, 11   |  resources:
 12     - |    limits:
 13     - |      cpu: 500m
 14     - |      memory: 1Gi
 15, 12   |    requests:
 16, 13   |      memory: 1Gi
 17, 14   |  source:
 18, 15   |    git:
...
  Really update the workload "spring-petclinic"? [yN]:
```

`--wait`

Holds the command until the workload is ready.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.5.0 --
type web --wait
  Update workload:
...
10, 10   |  source:
11, 11   |    git:
12, 12   |      ref:
13, 13   |        branch: main
    14 + |        tag: tap-1.5.0
14, 15   |      url: https://github.com/vmware-tanzu/application-accelerator-samples
15, 16   |    subPath: tanzu-java-web-app
  Really update the workload "tanzu-java-web-app"? Yes
  Updated workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

Waiting for workload "tanzu-java-web-app" to become ready...
Workload "tanzu-java-web-app" is ready
```

`--wait-timeout`

Sets a timeout to wait for the workload to become ready.

Example

```
tanzu apps workload apply tanzu-java-web-app --git-repo https://github.com/vmware-tanz
u/application-accelerator-samples --sub-path tanzu-java-web-app --git-tag tap-1.5.0-ta
ke1 --type web --wait --wait-timeout 1m
  Update workload:
...
10, 10   |  source:
11, 11   |    git:
12, 12   |      ref:
13, 13   |        branch: main
14     - |        tag: tap-1.5.0
    14 + |        tag: tap-1.5.0-take1
15, 15   |      url: https://github.com/vmware-tanzu/application-accelerator-samples
16, 16   |    subPath: tanzu-java-web-app
  Really update the workload "tanzu-java-web-app"? Yes
  Updated workload "tanzu-java-web-app"

To see logs:   "tanzu apps workload tail tanzu-java-web-app --timestamp --since 1h"
To get status: "tanzu apps workload get tanzu-java-web-app"

Waiting for workload "tanzu-java-web-app" to become ready...
Workload "tanzu-java-web-app" is ready
```

`--yes`, `-y`

Assumes `--yes` on all the survey prompts.

Example

```
tanzu apps workload apply spring-pet-clinic --local-path/home/user/workspace/spring-pe
t-clinic --source-image gcr.io/spring-community/spring-pet-clinic --type web -y
The files and/or directories listed in the .tanzuignore file are being excluded from t
he uploaded source code.
Publishing source in "/Users/dalfonso/Documents/src/java/tanzu-java-web-app" to "gcr.i
o/spring-community/spring-pet-clinic"...
  Published source

  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: spring-pet-clinic
      8 + |  namespace: default
      9 + |spec:
     10 + |  source:
     11 + |    image: gcr.io/spring-community/spring-pet-clinic:latest@sha256:5feb0d9d
af3f639755d8683ca7b647027cfddc7012e80c61dcdac27f0d7856a7
  Created workload "spring-pet-clinic"

To see logs:   "tanzu apps workload tail spring-pet-clinic --timestamp --since 1h"
To get status: "tanzu apps workload get spring-pet-clinic"
```

tanzu apps workload delete

This topic tells you about the Tanzu Apps CLI `tanzu apps workload delete` command. This command deletes workloads in a cluster. Deleting a workload does not mean the images published in the registry are deleted with it.

Default view

A message is displayed in the terminal asking if a workload should be deleted unless the `--yes` flag is used. If you indicate “Y”, then the workload starts a deletion process inside the cluster.

```
tanzu apps workload delete spring-pet-clinic
  Really delete the workload "spring-pet-clinic"? Yes
  Deleted workload "spring-pet-clinic"
```

```
tanzu apps workload delete spring-pet-clinic --yes
  Deleted workload "spring-pet-clinic"
```

Workload Delete flags

`--all`

Deletes all workloads in a namespace.

```
tanzu apps workload delete --all
  Really delete all workloads in the namespace "default"? (y/N) Y
  Deleted workloads in namespace "default"
```

```
tanzu apps workload delete --all -n my-namespace
  Really delete all workloads in the namespace "my-namespace"? Yes
  Deleted workloads in namespace "my-namespace"
```

`--file`, `-f`

Path to a file that contains the specification of the workload to be deleted.

```
tanzu apps workload delete -f path/to/file/spring-petclinic.yaml
  Really delete the workload "spring-petclinic"? Yes
  Deleted workload "spring-petclinic"
```

`--namespace`, `-n`

Specifies the namespace in which the workload is to be deleted.

```
tanzu apps workload delete spring-petclinic -n spring-petclinic-ns
  Really delete the workload "spring-petclinic"? Yes
  Deleted workload "spring-petclinic"
```

`wait`

Waits until workload is deleted.

```
tanzu apps workload delete -f path/to/file/spring-petclinic.yaml --wait
  Really delete the workload "spring-petclinic"? Yes
  Deleted workload "spring-petclinic"
Waiting for workload "spring-petclinic" to be deleted...
Workload "spring-petclinic" was deleted
```

`--wait-timeout`

Sets a timeout to wait for workload to be deleted.

```
tanzu apps workload delete -f path/to/file/spring-petclinic.yaml --wait --wait-timeout
1m
  Really delete the workload "spring-petclinic"? Yes
```

```
  Deleted workload "spring-petclinic"
Waiting for workload "spring-petclinic" to be deleted...
Workload "spring-petclinic" was deleted
```

```
tanzu apps workload delete spring-petclinic -n spring-petclinic-ns --wait --wait-timeo
ut 1m
  Really delete the workload "spring-petclinic"? Yes
  Deleted workload "spring-petclinic"
Waiting for workload "spring-petclinic" to be deleted...
Error: timeout after 1m waiting for "spring-petclinic" to be deleted
To view status run: tanzu apps workload get spring-petclinic --namespace spring-petcli
nic-ns
Error: exit status 1

✖  exit status 1
```

`--yes`, `-f`

Assume yes on all the survey prompts.

```
tanzu apps workload delete spring-petclinic --yes
  Deleted workload "spring-petclinic"
```

tanzu apps workload get

This topic tells you how to use the Tanzu Apps CLI `tanzu apps workload get` command to retrieve information and status about a workload.

Some of the workload details in the command output are as follows:

- Workload name, type, and namespace.
- The source code used to build the workload (or the pre-built OCI image).
- The supply chain that processed the workload.
- The specific resources within the supply chain that interacted with the workload, and the stamped out resources associated with each of those interactions.
- The delivery workflow that the application follows.
- Any issues associated with deploying the workload.
- The *pods* the workload generates.
- And when applicable, the knative services related to the workload.

Default view

There are multiple sections in the workload get command output. The following data is displayed:

- Name of the workload and its status.
- Displays source information of workload.
- If the workload was matched with a supply chain, the information of its name and the status is displayed.
- Information and status of the individual steps that is defined in the supply chain for the workload.
- Any issue with the workload: the name and corresponding message.

- Workload related resource information and status like services claims, related pods, knative services.

At the very end of the command output, a hint to follow up commands is also displayed.

Note

the `Supply Chain` and `Delivery` sections are included in the command output depending on whether those resources are present on the target cluster (e.g. If the target includes only build components, there would be no `Delivery` resources available and therefore the `Delivery` section would not be included in the command output.).

```
tanzu apps workload get rmq-sample-app
  Overview
   name:         rmq-sample-app
   type:         web
   namespace:    default

  Source
   type:      git
   url:       https://github.com/jhvhs/rabbitmq-sample
   branch:    main

  Supply Chain
   name:    source-to-url

   NAME                READY    HEALTHY    UPDATED    RESOURCE
   source-provider     True     True       7d11h      gitrepositories.source.toolkit.fluxc
d.io/rmq-sample-app
   image-provider      True     True       2d18h      images.kpack.io/rmq-sample-app
   config-provider     True     True       7d11h      podintents.conventions.carto.run/rmq
-sample-app
   app-config          True     True       7d11h      configmaps/rmq-sample-app
   service-bindings    True     True       7d11h      configmaps/rmq-sample-app-with-claim
s
   api-descriptors     True     True       7d11h      configmaps/rmq-sample-app-with-api-d
escriptors
   config-writer       True     True       2d18h      runnables.carto.run/rmq-sample-app-c
onfig-writer

  Delivery
   name:    delivery-basic

   NAME               READY      HEALTHY    UPDATED    RESOURCE
   source-provider    True       True       7d11h      imagerepositories.source.apps.tanz
u.vmware.com/rmq-sample-app-delivery
   deployer           True       True       6m25s      apps.kappctrl.k14s.io/rmq-sample-ap
p

  Messages
   No messages found.

  Services
   CLAIM    NAME                          KIND               API VERSION
   rmq      example-rabbitmq-cluster-1    RabbitmqCluster    rabbitmq.com/v1beta1

  Pods
   NAME                                     READY   STATUS      RESTARTS   AGE
   rmq-sample-app-build-1-build-pod         0/1     Completed   0          56d
   rmq-sample-app-build-2-build-pod         0/1     Completed   0          46d
   rmq-sample-app-build-3-build-pod         0/1     Completed   0          45d
```

```
   rmq-sample-app-config-writer-54mwk-pod   0/1     Completed   0          6d12h
   rmq-sample-app-config-writer-74qvp-pod   0/1     Completed   0          6d16h
   rmq-sample-app-config-writer-78r5w-pod   0/1     Completed   0          45d
   rmq-sample-app-config-writer-9xs5f-pod   0/1     Completed   0          46d

  Knative Services
   NAME             READY   URL
   rmq-sample-app   Ready   http://rmq-sample-app.default.127.0.0.1.nip.io

To see logs: "tanzu apps workload tail rmq-sample-app --timestamp --since 1h"
```

`--export`

Exports the submitted workload in `yaml` format. This flag can also be used with the `--output` flag. With export, the output is shortened because some text boxes are removed.

```
tanzu apps workload get tanzu-java-web-app --export

---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
labels:
    apps.tanzu.vmware.com/workload-type: web
    autoscaling.knative.dev/min-scale: "1"
name: tanzu-java-web-app
namespace: default
spec:
source:
    git:
    ref:
        tag: tap-1.3
      url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: tanzu-java-web-app
```

`--output`/`-o`

Configures how the workload is being shown. This supports the values `yaml`, `yml`, and `json`, where `yaml` and `yml` are equal. It shows the actual workload in the cluster.

- `yaml/yml`

  ```
  tanzu apps workload get tanzu-java-web-app -o yaml
  ---
  apiVersion: carto.run/v1alpha1
  kind: Workload
  metadata:
  creationTimestamp: "2022-06-03T18:10:59Z"
  generation: 1
  labels:
      apps.tanzu.vmware.com/workload-type: web
      autoscaling.knative.dev/min-scale: "1"
  ...
  spec:
  source:
      git:
          ref:
              tag: tap-1.1
          url: https://github.com/vmware-tanzu/application-accelerator-samples
      subPath: tanzu-java-web-app
  status:
      conditions:
  ```

```
    - lastTransitionTime: "2022-06-03T18:10:59Z"
        message: ""
        reason: Ready
        status: "True"
        type: SupplyChainReady
    - lastTransitionTime: "2022-06-03T18:14:18Z"
        message: ""
        reason: ResourceSubmissionComplete
        status: "True"
        type: ResourcesSubmitted
    - lastTransitionTime: "2022-06-03T18:14:18Z"
        message: ""
        reason: Ready
        status: "True"
        type: Ready
    observedGeneration: 1
    resources:
    ...
    supplyChainRef:
        kind: ClusterSupplyChain
        name: source-to-url
        ...
```

- json

```
tanzu apps workload get tanzu-java-web-app -o json
{
    "kind": "Workload",
    "apiVersion": "carto.run/v1alpha1",
    "metadata": {
        "name": "tanzu-java-web-app",
        "namespace": "default",
        "uid": "937679ca-9c72-4e23-bfef-6334e6c003a7",
        "resourceVersion": "111637840",
        "generation": 1,
        "creationTimestamp": "2022-06-03T18:10:59Z",
        "labels": {
            "apps.tanzu.vmware.com/workload-type": "web",
            "autoscaling.knative.dev/min-scale": "1"
        },
...
}
"spec": {
        "source": {
            "git": {
                "url": "https://github.com/vmware-tanzu/application-accelerator
-samples",
                "ref": {
                    "tag": "tap-1.3"
                }
            },
            "subPath": "tanzu-java-web-app"
        }
    },
    "status": {
        "observedGeneration": 1,
        "conditions": [
            {
                "type": "SupplyChainReady",
                "status": "True",
                "lastTransitionTime": "2022-06-03T18:10:59Z",
                "reason": "Ready",
                "message": ""
            },
            {
                "type": "ResourcesSubmitted",
```

```
                "status": "True",
                "lastTransitionTime": "2022-06-03T18:14:18Z",
                "reason": "ResourceSubmissionComplete",
                "message": ""
            },
            {
                "type": "Ready",
                "status": "True",
                "lastTransitionTime": "2022-06-03T18:14:18Z",
                "reason": "Ready",
                "message": ""
            }
        ],
        "supplyChainRef": {
            "kind": "ClusterSupplyChain",
            "name": "source-to-url"
        },
        "resources": [
            {
                "name": "source-provider",
                "stampedRef": {
                    "kind": "GitRepository",
                    "namespace": "default",
                    "name": "tanzu-java-web-app",
                    ...
                }
            }
        ]
        ...
    }
    ...
}
```

`--namespace`/`-n`

Specifies the namespace where the workload is deployed.

```
tanzu apps workload get tanzu-java-web-app -n development

  Overview
   name:        tanzu-java-web-app
   type:        web
   namespace:   development

  Source
   type:     git
   url:      https://github.com/vmware-tanzu/application-accelerator-samples
   sub-path: tanzu-java-web-app
   tag:      tap-1.3

  Supply Chain
   name:   source-to-url

   NAME               READY   HEALTHY   UPDATED   RESOURCE
   source-provider    True    True      31m       gitrepositories.source.toolkit.fluxc
d.io/tanzu-java-web-app
   image-provider     True    True      30m       images.kpack.io/tanzu-java-web-app
   config-provider    True    True      30m       podintents.conventions.carto.run/tan
zu-java-web-app
   app-config         True    True      30m       configmaps/tanzu-java-web-app
   service-bindings   True    True      30m       configmaps/tanzu-java-web-app-with-c
laims
   api-descriptors    True    True      30m       configmaps/tanzu-java-web-app-with-a
pi-descriptors
   config-writer      True    True      30m       runnables.carto.run/tanzu-java-web-a
```

```
pp-config-writer

  Delivery
   name:   delivery-basic

   NAME              READY   HEALTHY   UPDATED   RESOURCE
   source-provider   True    True      30m       imagerepositories.source.apps.tanzu.v
mware.com/tanzu-java-web-app-delivery
   deployer          True    True      30m       apps.kappctrl.k14s.io/tanzu-java-web-
app

  Messages
   No messages found.

  Pods
   NAME                                        READY   STATUS      RESTARTS   AGE
   tanzu-java-web-app-build-11-build-pod       0/1     Completed   0          6d12h
   tanzu-java-web-app-build-12-build-pod       0/1     Completed   0          22h
   tanzu-java-web-app-build-3-build-pod        0/1     Completed   0          60d
   tanzu-java-web-app-config-writer-655rb-pod  0/1     Completed   0          21d
   tanzu-java-web-app-config-writer-7h8bn-pod  0/1     Completed   0          6d12h
   tanzu-java-web-app-config-writer-7xr6m-pod  0/1     Completed   0          60d
   tanzu-java-web-app-config-writer-g9gp8-pod  0/1     Completed   0          45d

  Knative Services
   NAME                READY   URL
   tanzu-java-web-app  Ready   http://tanzu-java-web-app.default.127.0.0.1.nip.io

To see logs: "tanzu apps workload tail tanzu-java-web-app --namespace development --ti
mestamp --since 1h"
```

tanzu apps workload list

This topic tells you about the Tanzu Apps CLI `tanzu apps workload list` command.

The `tanzu apps workload list` command gets the workloads present in the cluster, either in the current namespace, in another namespace, or all namespaces.

Default view

The default view for workload list is a table with the workloads present in the cluster in the specified namespace. This table has, in each row, the name of the workload, the app it is related to, its status, and how long it's been in the cluster.

For example, in the default namespace

```
tanzu apps workload list

NAME                 TYPE      APP                 READY                    AGE
nginx4               web       <empty>             Ready                    7d9h
petclinic2           web       <empty>             Ready                    29h
rmq-sample-app       web       <empty>             Ready                    164m
rmq-sample-app4      web       <empty>             WorkloadLabelsMissing    29d
spring-pet-clinic    web       <empty>             Unknown                  166m
spring-petclinic2    web       spring-petclinic    Unknown                  29d
spring-petclinic3    <empty>   spring-petclinic    Ready                    29d
tanzu-java-web-app   web       tanzu-java-web-app  Ready                    40m
tanzu-java-web-app2  web       tanzu-java-web-app  Ready                    20m
```

>Workload List flags

`--all-namespaces`, `-A`

Shows workloads in all namespaces in cluster.

```
tanzu apps workload list -A

NAMESPACE   TYPE   NAME                  APP                  READY
AGE
default     web    nginx4                <empty>              Ready
7d9h
default     web    petclinic2            <empty>              Ready
30h
default     web    rmq-sample-app        <empty>              Ready
179m
default     web    rmq-sample-app4       <empty>              WorkloadLabelsMissing
29d
default     web    spring-pet-clinic     <empty>              Unknown
3h1m
default     web    spring-petclinic2     spring-petclinic     Unknown
29d
default     web    spring-petclinic3     spring-petclinic     Ready
29d
default     web    tanzu-java-web-app    tanzu-java-web-app   Ready
40m
default     web    tanzu-java-web-app2   tanzu-java-web-app   Ready
20m
nginx-ns    web    nginx2                <empty>              TemplateRejectedByAPISer
ver   8d
nginx-ns    web    nginx4                <empty>              TemplateRejectedByAPISer
ver   8d
```

`--app`

Shows workloads which app is the one specified in the command.

```
tanzu apps workload list --app spring-petclinic

NAME                TYPE   READY     AGE
spring-petclinic2   web    Unknown   29d
spring-petclinic3   web    Ready     29d
```

`--namespace`, `-n`

Lists all the workloads present in the specified namespace.

```
tanzu apps workload list -n my-namespace

NAME   TYPE   APP       READY                          AGE
app1   web    <empty>   TemplateRejectedByAPIServer   8d
app2   web    <empty>   Ready                          8d
app3   web    <empty>   Unknown                        8d
```

`--output`, `-o`

Allows to list all workloads in the specified namespace in yaml, yml or json format.

- yaml/yml

  ```
  ---
  - apiVersion: carto.run/v1alpha1
  kind: Workload
  metadata:
  ```

```
    creationTimestamp: "2022-05-17T22:06:49Z"
    generation: 1
    labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    managedFields:
    ...
    ...
    manager: cartographer
    operation: Update
    time: "2022-05-17T22:06:52Z"
name: tanzu-java-web-app2
namespace: default
resourceVersion: "6071972"
uid: 7fbcd40d-4eb3-41dc-a1db-657b64148708
spec:
    source:
        git:
            ref:
              tag: tap-1.3
            url: https://github.com/vmware-tanzu/application-accelerator-sample
s
        subPath: tanzu-java-web-app
...
...
---
- apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
    creationTimestamp: "2022-05-17T22:06:49Z"
    generation: 1
    labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    managedFields:
    ...
    ...
    manager: cartographer
    operation: Update
    time: "2022-05-17T22:06:52Z"
name: tanzu-java-web-app
namespace: default
resourceVersion: "6071972"
uid: 7fbcd40d-4eb3-41dc-a1db-657b64148708
spec:
    source:
        git:
            ref:
              tag: tap-1.3
            url: https://github.com/vmware-tanzu/application-accelerator-sample
s
        subPath: tanzu-java-web-app
...
...
```

- json

```
[
    {
        "kind": "Workload",
        "apiVersion": "carto.run/v1alpha1",
        "metadata": {
            "name": "tanzu-java-web-app2",
            "namespace": "default",
            "uid": "7fbcd40d-4eb3-41dc-a1db-657b64148708",
            "resourceVersion": "6071972",
```

```
            "generation": 1,
            "creationTimestamp": "2022-05-17T22:06:49Z",
            "labels": {
                "app.kubernetes.io/part-of": "tanzu-java-web-app",
                "apps.tanzu.vmware.com/workload-type": "web"
            },
        ...
        }
    ...
    },
    {
        "kind": "Workload",
        "apiVersion": "carto.run/v1alpha1",
        "metadata": {
            "name": "tanzu-java-web-app",
            "namespace": "default",
            "uid": "7fbcd40d-4eb3-41dc-a1db-657b64148708",
            "resourceVersion": "6071972",
            "generation": 1,
            "creationTimestamp": "2022-05-17T22:06:49Z",
            "labels": {
                "app.kubernetes.io/part-of": "tanzu-java-web-app",
                "apps.tanzu.vmware.com/workload-type": "web"
            },
        ...
        }
    ...
    },
...
...
]
```

tanzu apps workload tail

This topic tells you about the Tanzu Apps CLI `tanzu apps workload tail` command.

The `tanzu apps workload tail` checks the runtime logs of a workload.

Default view

Without timestamp set, workload tail will show the stage where it is and the related log.

```
+ spring-pet-clinic-build-1-build-pod › prepare
+ spring-pet-clinic-build-1-build-pod › detect
+ spring-pet-clinic-build-1-build-pod › analyze
+ spring-pet-clinic-build-1-build-pod › build
+ spring-pet-clinic-build-1-build-pod › restore
spring-pet-clinic-build-1-build-pod[detect] ======== Output: tanzu-buildpacks/poetry@
0.1.0 ========
spring-pet-clinic-build-1-build-pod[detect] pyproject.toml must include [tool.poetry.d
ependencies.python], see https://python-poetry.org/docs/pyproject/#dependencies-and-de
v-dependencies
spring-pet-clinic-build-1-build-pod[analyze] Restoring data for sbom from previous ima
ge
spring-pet-clinic-build-1-build-pod[detect] err:  tanzu-buildpacks/poetry@0.1.0 (1)
spring-pet-clinic-build-1-build-pod[detect] ======== Output: tanzu-buildpacks/poetry@
0.1.0 ========
spring-pet-clinic-build-1-build-pod[detect] pyproject.toml must include [tool.poetry.d
ependencies.python], see https://python-poetry.org/docs/pyproject/#dependencies-and-de
v-dependencies
spring-pet-clinic-build-1-build-pod[detect] err:  tanzu-buildpacks/poetry@0.1.0 (1)
spring-pet-clinic-build-1-build-pod[detect] 10 of 38 buildpacks participating
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/ca-certificates   3.1.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/bellsoft-liberica 9.2.0
```

```
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/syft               1.10.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/gradle             6.4.1
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/maven              6.4.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/executable-jar     6.1.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/apache-tomcat      7.2.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/dist-zip           5.2.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/spring-boot        5.8.0
spring-pet-clinic-build-1-build-pod[detect] paketo-buildpacks/image-labels       4.1.0
...
...
...
```

>Workload Tail flags

`--component`

Set the component from which the tail command should stream the logs. The values that the flag can take depend on the final deployed pods label `app.kubernetes.io/component`, for example, `build`, `run` and, `config-writer`

```
tanzu apps workload tail pet-clinic --component build

pet-clinic-build-1-build-pod[export] Adding label 'io.buildpacks.project.metadata'
pet-clinic-build-1-build-pod[export] Adding label 'org.opencontainers.image.title'
pet-clinic-build-1-build-pod[export] Adding label 'org.opencontainers.image.version'
pet-clinic-build-1-build-pod[export] Adding label 'org.springframework.boot.version'
pet-clinic-build-1-build-pod[export] Adding label 'org.opencontainers.image.source'
pet-clinic-build-1-build-pod[export] Setting default process type 'web'
pet-clinic-build-1-build-pod[export] Saving gcr.io/dalfonso-tanzu-dev-frmwrk/pet-clini
c-default...
pet-clinic-build-1-build-pod[export] *** Images (sha256:2ae6154c4433d870a330a0c2fc8253
40c3ead2603e3d1526e47c47cb6297fffe):
pet-clinic-build-1-build-pod[export]       gcr.io/dalfonso-tanzu-dev-frmwrk/pet-clinic
-default
pet-clinic-build-1-build-pod[export]       gcr.io/dalfonso-tanzu-dev-frmwrk/pet-clinic
-default:b1.20220603.181107
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/bellsoft-li
berica:jdk'
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/syft:syft'
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/maven:appli
cation'
pet-clinic-build-1-build-pod[export] Adding cache layer 'paketo-buildpacks/maven:cach
e'
pet-clinic-build-1-build-pod[export] Adding cache layer 'cache.sbom'
```

`--namespace, -n`

Specifies the namespace where the workload was deployed to get logs from.

```
tanzu apps workload tail pet-clinic -n development

pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:52.684  INFO 1
--- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engi
ne: [Apache Tomcat/9.0.63]
+ pet-clinic-build-3-build-pod › export
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:52.699  INFO 1
--- [           main] o.a.c.c.C.[Tomcat-1].[localhost].[/]     : Initializing Spring e
mbedded WebApplicationContext
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:52.699  INFO 1
--- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationCo
ntext: initialization completed in 131 ms
pet-clinic-00004-deployment-6445565f7b-ts815[workload] 2022-06-14 16:28:52.755  INFO 1
```

```
--- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 13 endpoint
(s) beneath base path '/actuator'
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.059  INFO 1
--- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on por
t(s): 8081 (http) with context path ''
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.074  INFO 1
--- [           main] o.s.s.petclinic.PetClinicApplication     : Started PetClinicAppl
ication in 8.373 seconds (JVM running for 8.993)
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.229  INFO 1
--- [nio-8081-exec-1] o.a.c.c.C.[Tomcat-1].[localhost].[/]     : Initializing Spring D
ispatcherServlet 'dispatcherServlet'
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.229  INFO 1
--- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet
'dispatcherServlet'
pet-clinic-00004-deployment-6445565f7b-ts8l5[workload] 2022-06-14 16:28:53.231  INFO 1
--- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initializat
ion in 2 ms
```

--since

Sets the time duration to start reading logs from, this is set in seconds (`s`), minutes(`m`) or hours (`h`) in the format `0h0m0s`. You do not need to indicate a `0` duration, for example, 1 hour, 0 minutes and 1 seconds is `1h1s`. The default value is 1 second `1s`

```
tanzu apps workload tail pet-clinic --since 1h1s

pet-clinic-config-writer-9fbk6-pod[place-tools] 2022/06/14 16:28:04 Copied /ko-app/ent
rypoint to /tekton/bin/entrypoint
pet-clinic-config-writer-9fbk6-pod[place-scripts] 2022/06/14 16:28:06 Decoded script /
tekton/scripts/script-0-dz84w
pet-clinic-config-writer-9fbk6-pod[step-init] 2022/06/14 16:28:05 Setup /step director
ies
pet-clinic-config-writer-9fbk6-pod[step-main] ++ mktemp -d
pet-clinic-config-writer-9fbk6-pod[step-main] + cd /tmp/tmp.n4ObHYVxpl
pet-clinic-config-writer-9fbk6-pod[step-main] + echo -e eyJkZWxpdmVyeS55bWwiOiJhcGlWZX
JzaW9uOiBzZXJ2aW5nLmtuYXRpdmUuZGV2L3YxXG5raW5kOiBTZXJ2aWNlXG5tZXRhZGF0YTpcbiAgbmFtZTog
cGV0LWNsaW5pY1xuICBsYWJlbHM6XG4gICAgYXBwcy50YW56dS52bXdhcmUuY29tL3dvcmtsb2FkLXR5cGU6IH
dlYlxuICAgIGF1dG9zY2FsaW5nLmtuYXRpdmUuZGV2L21pbi1zY2FsZTogXCIxXCJcbiAgICBhcHAua3ViZXJu
ZXRlcy5pby9jb21wb25lbnQ6IHJ1blxuICAgIGNhcnRvLnJ1bi93b3JrbG9hZC1uYW1lOiBwZXQtY2xpbmljXG
5zcGVjOlxuICB0ZW1wbGF0ZTpcbiAgICBtZXRhZGF0YTpcbiAgICAgIGFubm90YXRpb25zOlxuICAgICAgICBi
b290LnNwcmluZy5pby9hY3R1YXRvcjogaHR0cDovLzo4MDgxL2FjdHVhdG9yXG4gICAgICAgIGJvb3Quc3ByaW
5nLmlvL3ZlcnNpb246IDIuNi44XG4gICAgICAgIGNvbnZlbnRpb25zLmFwcHMudGFuenUudm13YXJlLmNvbS9h
cHBsaWVkLWNvbnZlbnRpb25zOiB8LVxuICAgICAgICAgIHNwcmluZy1ib290LWNvbnZlbnRpb24vc3ByaW5nLW
Jvb3RcbiAgICAgICAgICBzcHJpbmctYm9vdC1jb252ZW50aW9uL3NwcmluZy1ib290LWdyYWNlZnVsLXNodXRk
b3duXG4gICAgICAgICAgc3ByaW5nLWJvb3QtY29udmVudGlvbi9zcHJpbmctYm9vdC13ZWJcbiAgICAgICAgIC
BzcHJpbmctYm9vdC1jb252ZW50aW9uL3NwcmluZy1ib290LWFjdHVhdG9yXG4gICAgICAgICAgc3ByaW5nLWJv
b3QtY29udmVudGlvbi9zcHJpbmctYm9vdC1hY3R1YXRvci1wcm9iZXNcbiAgICAgICAgICBzcHJpbmctYm9vdC
1jb252ZW50aW9uL3NlcnZpY2UtaW50ZW50LW15c3FsXG4gICAgICAgICAgc3ByaW5nLWJvb3QtY29udmVudGlv
bi9zZXJ2aWNlLWludGVudC1wb3N0Z3Jlc1xuICAgICAgICAgIGFwcGxpdmV2aWV3LXNhbXBsZS9hcHAtbGl2ZS
12aWV3LWNvbm5lY3RvclxuICAgICAgICAgIGFwcGxpdmV2aWV3LXNhbXBsZS9hcHAtbGl2ZS12aWV3LWFwcGZs
YXZvdXJzXG4gICAgICAgICAgYXBwbGl2ZXZpZXctc2FtcGxlL2FwcC1saXZlLXZpZXctc3lzdGVtcHJvcGVydG
llc1xuICAgICAgICBkZXZlbG9wZXIuY29udmVudGlvbnMvdGFyZ2V0LWNvbnRhaW5lcnM6IHdvcmtsb2FkXG4g
ICAgICAgIHNlcnZpY2VzLmNvbnZlbnRpb25zLmFwcHMudGFuenUudm13YXJlLmNvbS9teXNxbDogbXlzcWwtY2
9ubmVjdG9yLWphdmEvOC4wLjI5XG4gICAgICAgIHNlcnZpY2VzLmNvbnZlbnRpb25zLmFwcHMudGFuenUudm13
YXJlLmNvbS9wb3N0Z3JlczogcG9zdGdyZXNxbC80Mi4zLjVcbiAgICAgIGxhYmVsczpcbiAgICAgICAgYXBwLm
t1YmVybmV0ZXMuaW8vY29tcG9uZW50OiBydW5cbiAgICAgICAgYXBwcy50YW56dS52bXdhcmUuY29tL3dvcmts
b2FkLXR5cGU6IHdlYlxuICAgICAgICBjYXJ0by5ydW4vd29ya2xvYWQtbmFtZTogcGV0LWNsaW5pY1xuICAgIC
AgICBjb252ZW50aW9ucy5hcHBzLnRhbnp1LnZtd2FyZS5jb20vZnJhbWV3b3JrOiBzcHJpbmctYm9vdFxuICAg
ICAgICBzZXJ2aWNlcy5jb252ZW50aW9ucy5hcHBzLnRhbnp1LnZtd2FyZS5jb20vbXlzcWw6IHdvcmtsb2FkXG
4gICAgICAgIHNlcnZpY2VzLmNvbnZlbnRpb25zLmFwcHMudGFuenUudm13YXJlLmNvbS9wb3N0Z3Jlczogd29y
a2xvYWRcbiAgICAgICAgdGFuenUuYXBwLmxpdmUudmlldzogXCJ0cnVlXCJcbiAgICAgICAgdGFuenUuYXBwLm
xpdmUudmlldy5hcHBsaWNhdGlvbi5hY3R1YXRvci5wb3J0OiBcIjgwODFcIlxuICAgICAgICB0YW56dS5hcHAu
bGl2ZS52aWV3LmFwcGxpY2F0aW9uLmZsYXZvdXJzOiBzcHJpbmctYm9vdFxuICAgICAgICB0YW56dS5hcHAubG
l2ZS52aWV3LmFwcGxpY2F0aW9uLm5hbWU6IHBldGNsaW5pY1xuICAgIHNwZWM6XG4gICAgICBjb250YWluZXJz
```

```
OlxuICAgICAgLSBlbnY6XG4gICAgICAgIC0gbmFtZTogSkFWQV9UT09MX09QVElPTlNcbiAgICAgICAgICB2YW
x1ZTogLURtYW5hZ2VtZW50LmVuZHBvaW50LmhlYWx0aC5wcm9iZXMuYWRkLWFkZGl0aW9uYWwtcGF0aHM9XCJ0
cnVlXCIgLURtYW5hZ2VtZW50LmVuZHBvaW50LmhlYWx0aC5zaG93LWRldGFpbHM9YWx3YXlzIC1EbWFuYWdlbW
VudC5lbmRwb2ludHMud2ViLmJhc2UtcGF0aD1cIi9hY3R1YXRvclwiIC1EbWFuYWdlbWVudC5lbmRwb2ludHMu
d2ViLmV4cG9zdXJlLmluY2x1ZGU9KiAtRG1hbmFnZW1lbnQuaGVhbHRoLnByb2Jlcy5lbmFibGVkPVwidHJ1ZV
wiIC1EbWFuYWdlbWVudC5zZXJ2ZXIucG9ydD1cIjgwODFcIiAtRHNlcnZlci5wb3J0PVwiODA4MFwiIC1Ec2Vy
dmVyLnNodXRkb3duLmdyYWNlLXBlcmlvZD1cIjI0c1wiXG4gICAgICAgIGltYWdlOiBnY3IuaW8vZGFsZm9uc2
8tdGFuenUtZGV2LWZybXdyay9wZXQtY2xpbmljLWRlZmF1bHRAc2hhMjU2OjM5NjRiNTQwNTVlZjNkNmFiNWQ3
YTM5MmVjOGU3OWJhOTg2NjczODU2NmIyOGE2OGY4ZDM2YWY5YjkyMGJhODNcbiAgICAgICAgbGl2ZW5lc3NQcm
9iZTpcbiAgICAgICAgICBodHRwR2V0OlxuICAgICAgICAgICAgcGF0aDogL2xpdmV6XG4gICAgICAgICAgICBw
b3J0OiA4MDgwXG4gICAgICAgICAgICBzY2hlbWU6IEhUVFBBcbiAgICAgICAgbmFtZTogd29ya2xvYWRcbiAgIC
AgICAgcG9ydHM6XG4gICAgICAgIC0gY29udGFpbmVyUG9ydDogODA4MFxuICAgICAgICAgIHByb3RvY29sOiBU
Q1BBcbiAgICAgICAgcmVhZGluZXNzUHJvYmU6XG4gICAgICAgICAgaHR0cEdldDpcbiAgICAgICAgICAgIHBhdG
g6IC9yZWFkeXpcbiAgICAgICAgICAgIHBvcnQ6IDgwODBcbiAgICAgICAgICAgIHNjaGVtZTogSFRUUFxuICAg
ICAgICByZXNvdXJjZXM6IHt9XG4gICAgICAgIHNlY3VyaXR5Q29udGV4dDpcbiAgICAgICAgICBydW5Bc1VzZX
I6IDEwMDBcbiAgICAgIHNlcnZpY2VBY2NvdW50TmFtZTogZGVmYXVsdFxuIn0=
pet-clinic-config-writer-9fbk6-pod[step-main] + base64 --decode
pet-clinic-config-writer-9fbk6-pod[step-main] ++ cat files.json
+ pet-clinic-config-writer-kpmc6-pod › place-tools
pet-clinic-config-writer-9fbk6-pod[step-main] ++ jq -r 'to_entries | .[] | @sh "mkdir
-p $(dirname \(.key)) && echo \(.value) > \(.key)"'
+ pet-clinic-config-writer-kpmc6-pod › step-main
+ pet-clinic-config-writer-kpmc6-pod › step-init
+ pet-clinic-config-writer-kpmc6-pod › place-scripts
pet-clinic-config-writer-9fbk6-pod[step-main] + eval 'mkdir -p $(dirname '\''delivery.
yml'\'') && echo '\''apiVersion: serving.knative.dev/v1'
pet-clinic-config-writer-9fbk6-pod[step-main] kind: Service
pet-clinic-config-writer-9fbk6-pod[step-main] metadata:
pet-clinic-config-writer-9fbk6-pod[step-main]   name: pet-clinic
pet-clinic-config-writer-9fbk6-pod[step-main]   labels:
pet-clinic-config-writer-9fbk6-pod[step-main]     apps.tanzu.vmware.com/workload-type:
web
pet-clinic-config-writer-9fbk6-pod[step-main]     autoscaling.knative.dev/min-scale:
"1"
pet-clinic-config-writer-9fbk6-pod[step-main]     app.kubernetes.io/component: run
pet-clinic-config-writer-9fbk6-pod[step-main]     carto.run/workload-name: pet-clinic
```

`--timestamp`, `-t`

Adds the timestamp to the beginning of each log message

```
tanzu apps workload tail pet-clinic -t

pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645910625-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645942876-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645951930-0
5:00               |\      _,,,--,,_
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645957151-0
5:00              /,`.-'`'   ._  \-;;,_
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645961411-0
5:00   _______ __|,4-  ) )_   .;.(__`'-'__     ___ __    _ ___ _______
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645967316-0
5:00  |       | '---''(_/._)-'(_\_)   |   |   |   |  | |   |       |
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645971010-0
5:00  |    _  |    ___|_     _|       |   |   |   |   |_| |   |       | __ _ _
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645976591-0
5:00  |   |_| |   |___  |   | |       |   |   |   |       |   |       | \ \ \ \
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645986474-0
5:00  |    ___|    ___| |   | |      _|   |___|   |  _    |   |      _|  \ \ \ \
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645990521-0
5:00  |   |   |   |___  |   | |     |_|       |   | | |   |   |     |_    ) ) ) )
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645994112-0
5:00  |___|   |_______| |___| |_______|_______|___|_|  |__|___|_______|  / / / /
```

```
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.645998053-0
5:00  =========================================================================/_/_/_/
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.646001577-0
5:00
pet-clinic-00002-deployment-5cc69cfdc8-t45sc[workload] 2022-06-09T18:10:07.646005296-0
5:00 :: Built with Spring Boot :: 2.6.8
```

Tanzu Accelerator CLI overview

The Tanzu Accelerator Tanzu CLI includes commands for developers and operators to create and use accelerators.

Server API connections for operators and developers

The Tanzu Accelerator CLI must connect to a server for all provided commands except for the `help` and `version` commands.

Operators typically use **create**, **update**, and **delete** commands for managing accelerators in a Kubernetes context. They also use the **fragment** commands to manage accelerator fragments. These commands require a Kubernetes context where the operator is already authenticated and is authorized to create and edit the accelerator resources. Operators can also use the **get** and **list** commands by using the same authentication. For any of these commands, the operator can specify the `--context` flag to access accelerators in a specific Kubernetes context.

Developers use the **list**, **get**, and **generate** commands for using accelerators available in an Application Accelerator server. Developers use the `--server-url` to point to the Application Accelerator server they want to use.

You can either use the proxy that is part of TAP-GUI or you can use the URL for the Application Accelerator server, if that is configured to be exposed. VMware recommends using the TAP-GUI address.

Using TAP-GUI URL

1. Specify `--server-url` as:

   ```
   https://tap-gui.DOMAIN
   ```

   Where `DOMAIN` defaults to the `shared.ingress_domain` value provided in the Tanzu Application Platform values file.

2. Add the following flags to the `tap-values.yaml` file when `shared.ingress_domain` is set.

   ```
   accelerator:
     ingress:
       include: true
   ```

Using Application Accelerator Server URL

If you cannot use the TAP-GUI URL, the fallback is to use Application Accelerator server directly. In this case the URL depends on the configuration settings for Application Accelerator:

- For installations configured with a **shared ingress** and where the Application Accelerator server is configured to use the ingress, use `https://accelerator.<domain>` where `domain` defaults to the `shared.ingress_domain` value provided in the values file of Tanzu Application Platform.
- For installations using a **LoadBalancer**, look up the External IP address by using:

```
kubectl get -n accelerator-system service/acc-server
```

Use `http://<External-IP>` as the URL.

- For any other configuration, you can use port forwarding by using:

  ```
  kubectl port-forward service/acc-server -n accelerator-system 8877:80
  ```

  Use `http://localhost:8877` as the URL.

Using “ACC_SERVER_URL” environment variable

The developer can set an `ACC_SERVER_URL` environment variable to avoid having to provide the same `--server-url` flag for every command. Run `export ACC_SERVER_URL=<URL>` for the terminal session in use. If the developer explicitly specifies the `--server-url` flag, it overrides the `ACC_SERVER_URL` environment variable if it is set.

Installation

For information about installing the Tanzu CLI Accelerator plug-in, see Install Accelerator CLI plug-in.

Command reference

For information about available commands, see Command Reference.

Tanzu Accelerator CLI overview

The Tanzu Accelerator Tanzu CLI includes commands for developers and operators to create and use accelerators.

Server API connections for operators and developers

The Tanzu Accelerator CLI must connect to a server for all provided commands except for the `help` and `version` commands.

Operators typically use **create**, **update**, and **delete** commands for managing accelerators in a Kubernetes context. They also use the **fragment** commands to manage accelerator fragments. These commands require a Kubernetes context where the operator is already authenticated and is authorized to create and edit the accelerator resources. Operators can also use the **get** and **list** commands by using the same authentication. For any of these commands, the operator can specify the `--context` flag to access accelerators in a specific Kubernetes context.

Developers use the **list**, **get**, and **generate** commands for using accelerators available in an Application Accelerator server. Developers use the `--server-url` to point to the Application Accelerator server they want to use.

You can either use the proxy that is part of TAP-GUI or you can use the URL for the Application Accelerator server, if that is configured to be exposed. VMware recommends using the TAP-GUI address.

Using TAP-GUI URL

1. Specify `--server-url` as:

   ```
   https://tap-gui.DOMAIN
   ```

Where `DOMAIN` defaults to the `shared.ingress_domain` value provided in the Tanzu Application Platform values file.

2. Add the following flags to the `tap-values.yaml` file when `shared.ingress_domain` is set.

```
accelerator:
  ingress:
    include: true
```

Using Application Accelerator Server URL

If you cannot use the TAP-GUI URL, the fallback is to use Application Accelerator server directly. In this case the URL depends on the configuration settings for Application Accelerator:

- For installations configured with a **shared ingress** and where the Application Accelerator server is configured to use the ingress, use `https://accelerator.<domain>` where `domain` defaults to the `shared.ingress_domain` value provided in the values file of Tanzu Application Platform.
- For installations using a **LoadBalancer**, look up the External IP address by using:

  ```
  kubectl get -n accelerator-system service/acc-server
  ```

  Use `http://<External-IP>` as the URL.
- For any other configuration, you can use port forwarding by using:

  ```
  kubectl port-forward service/acc-server -n accelerator-system 8877:80
  ```

  Use `http://localhost:8877` as the URL.

Using “ACC_SERVER_URL” environment variable

The developer can set an `ACC_SERVER_URL` environment variable to avoid having to provide the same `--server-url` flag for every command. Run `export ACC_SERVER_URL=<URL>` for the terminal session in use. If the developer explicitly specifies the `--server-url` flag, it overrides the `ACC_SERVER_URL` environment variable if it is set.

Installation

For information about installing the Tanzu CLI Accelerator plug-in, see Install Accelerator CLI plug-in.

Command reference

For information about available commands, see Command Reference.

Install Tanzu Accelerator CLI

This topic tells you how to install the Tanzu Accelerator CLI.

Note

Follow the steps in this topic if you do not want to use a profile to install Tanzu Accelerator CLI. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before you install the Tanzu Accelerator CLI:

- Follow the instructions to Install or update the Tanzu CLI and plug-ins.

Install

To install the Tanzu Accelerator CLI:

1. From the `$HOME/tanzu` directory, run:

   ```
   tanzu plugin install accelerator
   ```

2. To verify that the CLI is installed correctly, run:

   ```
   tanzu accelerator version
   ```

   A version will be displayed in the output.

   If the following error is displayed during installation:

   ```
   Error: could not find plug-in "accelerator" in any known repositories

   ✖  could not find plug-in "accelerator" in any known repositories
   ```

   Verify that there is an `accelerator` entry in the `cli/manifest.yaml` file:

   ```
   plugins:
   ...
       - name: accelerator
       description: Manage accelerators in a Kubernetes cluster
       versions: []
   ```

Command reference

This topic provides you with a list of the Tanzu Accelerator CLI commands.

- tanzu accelerator
  - tanzu accelerator apply
  - tanzu accelerator create
  - tanzu accelerator delete
  - tanzu accelerator fragment
  - tanzu accelerator fragment create
  - tanzu accelerator fragment delete
  - tanzu accelerator fragment get
  - tanzu accelerator fragment list
  - tanzu accelerator fragment update
  - tanzu accelerator generate
  - tanzu accelerator generate-from-local
  - tanzu accelerator get
  - tanzu accelerator list

Command reference

This topic provides you with a list of the Tanzu Accelerator CLI commands.

tanzu accelerator

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator` command to manage accelerators in a Kubernetes cluster.

Options

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
  -h, --help              help for accelerator
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator apply - Apply accelerator resource
- tanzu accelerator create - Create a new accelerator
- tanzu accelerator delete - Delete an accelerator
- tanzu accelerator fragment create - Create a fragment
- tanzu accelerator generate - Generate project from accelerator
- tanzu accelerator generate-from-local - Generate a project from local or registered accelerators/fragments

- tanzu accelerator get - Get accelerator information
- tanzu accelerator list - List accelerators
- tanzu accelerator push - Push local path to source image
- tanzu accelerator update - Update an accelerator

tanzu accelerator

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator` command to manage accelerators in a Kubernetes cluster.

Options

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
  -h, --help              help for accelerator
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator apply - Apply accelerator resource
- tanzu accelerator create - Create a new accelerator
- tanzu accelerator delete - Delete an accelerator
- tanzu accelerator fragment create - Create a fragment
- tanzu accelerator generate - Generate project from accelerator
- tanzu accelerator generate-from-local - Generate a project from local or registered accelerators/fragments
- tanzu accelerator get - Get accelerator information
- tanzu accelerator list - List accelerators
- tanzu accelerator push - Push local path to source image
- tanzu accelerator update - Update an accelerator

tanzu accelerator apply

tanzu accelerator apply

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator apply` command to create or update accelerators.

Synopsis

Create or update accelerator resource using specified manifest file.

```
tanzu accelerator apply [flags]
```

Examples

```
tanzu accelerator apply --filename <path-to-resource-manifest>
```

Options

```
  -f, --filename string    path of manifest file for the resource
  -h, --help               help for apply
  -n, --namespace string   namespace for the resource (default "accelerator-system")
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator create

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator create` command to create a new accelerator.

Synopsis

Create a new accelerator resource with specified configuration.

Accelerator configuration options include:

- Git repository URL and branch/tag where accelerator code and metadata is defined
- Metadata like description, display-name, tags, and icon-url

The Git repository option is required. Metadata options are optional and override any values for the same options specified in the accelerator metadata retrieved from the Git repository.

```
tanzu accelerator create [flags]
```

Examples

```
tanzu accelerator create <accelerator-name> --git-repository <URL> --git-branch <branc
h>
```

Options

```
      --description string    description of this accelerator
      --display-name string   display name for the accelerator
      --git-branch string     Git repository branch to be used (default "main")
      --git-repo string       Git repository URL for the accelerator
      --git-sub-path string   Git repository subPath to be used
      --git-tag string        Git repository tag to be used
  -h, --help                  help for create
      --icon-url string       URL for icon to use with the accelerator
      --interval string       interval for checking for updates to Git or image reposi
tory
      --local-path string     path to the directory containing the source for the acce
lerator
  -n, --namespace string      namespace for accelerator system (default "accelerator-s
ystem")
      --secret-ref string     name of secret containing credentials for private Git or
```

```
image repository
      --source-image string   name of the source image for the accelerator
      --tags strings          tags that can be used to search for accelerators
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

tanzu accelerator

tanzu accelerator delete

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator delete` command to delete an accelerator.

Synopsis

Delete the accelerator resource with the specified name.

```
tanzu accelerator delete [flags]
```

Examples

```
tanzu accelerator delete <accelerator-name>
```

Options

```
  -h, --help               help for delete
  -n, --namespace string   namespace for accelerator system (default "accelerator-syst
em")
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator fragment

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment` command to manage fragments.

Synopsis

Commands to manage accelerator fragments

Examples

```
tanzu accelerator fragment --help
```

Options

```
  -h, --help   help for fragment
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- Using accelerator fragments
- tanzu accelerator - Manage accelerators in a Kubernetes cluster
- tanzu accelerator fragment create - Create a new accelerator fragment
- tanzu accelerator fragment delete - Delete an accelerator fragment
- tanzu accelerator fragment get - Get accelerator fragment information
- tanzu accelerator fragment list - List accelerator fragments
- tanzu accelerator fragment update - Update an accelerator fragment

tanzu accelerator fragment create

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment create` command to create a new accelerator fragment.

Synopsis

Create a new accelerator fragment resource with specified configuration.

Accelerator configuration options include:

- Git repository URL and branch/tag where accelerator code and metadata is defined
- Metadata like description, display-name, tags and icon-url

The Git repository option is required. Metadata options are optional and will override any values for the same options specified in the accelerator metadata retrieved from the Git repository.

```
tanzu accelerator fragment create [flags]
```

Example

```
tanzu accelerator fragment create <fragment-name> --git-repository <URL> --git-branch
<branch> --git-sub-path <sub-path>
```

Options

```
      --display-name string   display name for the accelerator fragment
      --git-branch string     Git repository branch to be used (default "main")
      --git-repo string       Git repository URL for the accelerator fragment
      --git-sub-path string   Git repository subPath to be used
      --git-tag string        Git repository tag to be used
  -h, --help                  help for create
      --interval string       interval for checking for updates to Git or image reposi
tory
      --local-path string     path to the directory containing the source for the acce
lerator fragment
  -n, --namespace string      namespace for accelerator system (default "accelerator-s
ystem")
      --secret-ref string     name of secret containing credentials for private Git or
image repository
      --source-image string   name of the source image for the accelerator
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

tanzu accelerator fragment

tanzu accelerator fragment delete

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment delete` command to delete an accelerator fragment.

Synopsis

Delete the accelerator fragment resource with the specified name.

```
tanzu accelerator fragment delete [flags]
```

Examples

```
tanzu accelerator fragment delete <fragment-name>
```

Options

```
  -h, --help               help for delete
  -n, --namespace string   namespace for accelerator system (default "accelerator-syst
em")
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
```

```
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

tanzu accelerator fragment

tanzu accelerator fragment get

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment get` command to get accelerator fragment information.

Synopsis

Get accelerator fragment information.

```
tanzu accelerator fragment get [flags]
```

Examples

```
tanzu accelerator get <fragment-name>
```

Options

```
  -h, --help               help for get
  -n, --namespace string   namespace for accelerator system (default "accelerator-syst
em")
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator fragment - Fragment commands

tanzu accelerator fragment list

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment list` to list accelerator fragments.

Synopsis

List all accelerator fragments.

```
tanzu accelerator fragment list [flags]
```

Examples

```
tanzu accelerator fragment list
```

Options

```
  -h, --help               help for list
  -n, --namespace string   namespace for accelerator system (default "accelerator-syst
em")
  -v, --verbose            include repository and show long URLs or image digests in t
he output
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

tanzu accelerator fragment

tanzu accelerator fragment update

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator fragment update` command to update an accelerator fragment.

Synopsis

Update an accelerator fragment resource with the specified name using the specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like display-name

The update command also provides a –reconcile flag that will force the accelerator fragment to be refreshed with any changes made to the associated Git repository.

```
tanzu accelerator fragment update [flags]
```

Examples

```
tanzu accelerator update <accelerator-name> --description "Lorem Ipsum"
```

Options

```
      --display-name string   display name for the accelerator fragment
      --git-branch string     Git repository branch to be used
      --git-repo string       Git repository URL for the accelerator fragment
      --git-sub-path string   Git repository subPath to be used
      --git-tag string        Git repository tag to be used
  -h, --help                  help for update
      --interval string       interval for checking for updates to Git repository
  -n, --namespace string      namespace for accelerator fragments (default "accelerato
r-system")
      --reconcile             trigger a reconciliation including the associated GitRep
ository resource
      --secret-ref string     name of secret containing credentials for private Git re
pository
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator fragment - Fragment commands

tanzu accelerator generate

tanzu accelerator generate

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator generate` command to generate a project from an accelerator.

Synopsis

Generate a project from an accelerator using provided options and download project artifacts as a ZIP file.

Generation options are provided as a JSON string and should match the metadata options that are specified for the accelerator used for the generation. The options can include “projectName” which defaults to the name of the accelerator. This “projectName” will be used as the name of the generated ZIP file.

You can see the available options by using the “tanzu accelerator get ” command.

Here is an example of an options JSON string that specifies the “projectName” and an “includeKubernetes” boolean flag:

```
--options '{"projectName":"test", "includeKubernetes": true}'
```

You can also provide a file that specifies the JSON string using the –options-file flag.

The generate command needs access to the Application Accelerator server. You can specify the –server-url flag or set an ACC_SERVER_URL environment variable. If you specify the –server-url flag it overrides the ACC_SERVER_URL environment variable if it is set.

```
tanzu accelerator generate [flags]
```

Examples

```
tanzu accelerator generate <accelerator-name> --options '{"projectName":"test"}'
```

Options

```
  -h, --help                  help for generate
      --options string        options JSON string
      --options-file string   path to file containing options JSON string
      --output-dir string     directory that the zip file will be written to
      --server-url string     the URL for the Application Accelerator server
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator generate-from-local

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator generate-from-local` command to generate a project from a combination of registered and local artifacts.

Synopsis

Generate a project from a combination of local files and registered accelerators/fragments using provided options and download project artifacts as a ZIP file.

Options values are provided as a JSON object and should match the declared options that are specified for the accelerator used for the generation. The options can include “projectName” which defaults to the name of the accelerator. This “projectName” is used as the name of the generated ZIP file.

Here is an example of an options JSON string that specifies the “projectName” and an “includeKubernetes” Boolean flag:

```
--options '{"projectName":"test", "includeKubernetes": true}'
```

You can also provide a file that specifies the JSON string using the –options-file flag.

The generate-from-local command needs access to the Application Accelerator server. You can specify the –server-url flag or set an ACC_SERVER_URL environment variable. If you specify the –server-url flag it overrides the ACC_SERVER_URL environment variable if it is set.

```
tanzu accelerator generate-from-local [flags]
```

Examples

```
tanzu accelerator generate-from-local --accelerator-path java-rest=workspace/java-rest
--fragment-paths java-version=workspace/version --fragment-names tap-workload --option
s '{"projectName":"test"}'
```

Options

```
      --accelerator-name string             name of the registered accelerator to use
      --accelerator-path "key=value" pair   key value pair of the name and path to the
directory containing the accelerator
  -f, --force                               force clean and rewrite of output-dir
      --fragment-names strings              names of the registered fragments to use
      --fragment-paths stringToString       key value pairs of the name and path to th
e directory containing each fragment (default [])
  -h, --help                                help for generate-from-local
      --options string                      options JSON string (default "{}")
      --options-file string                 path to file containing options JSON strin
g
  -o, --output-dir string                   the directory that the project will be cre
ated in (defaults to the project name)
```

```
      --server-url string                   the URL for the Application Accelerator se
rver
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

tanzu accelerator

tanzu accelerator get

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator get` command to get accelerator information.

Synopsis

Get accelerator information.

You can choose to get the accelerator from the Application Accelerator server using –server-url flag or from a Kubernetes context using –from-context flag. The default is to get accelerators from the Kubernetes context. To override this, you can set the ACC_SERVER_URL environment variable with the URL for the Application Accelerator server you want to access.

```
tanzu accelerator get [flags]
```

Examples

```
tanzu accelerator get <accelerator-name> --from-context
```

Options

```
      --from-context        retrieve resources from current context defined in kubecon
fig
  -h, --help                help for get
  -n, --namespace string    namespace for accelerator system (default "accelerator-sys
tem")
      --server-url string   the URL for the Application Accelerator server
  -v, --verbose             include all fields and show long URLs in the output
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

tanzu accelerator

tanzu accelerator list

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator list` command to list accelerators.

Synopsis

List all accelerators.

You can choose to list the accelerators from the Application Accelerator server using –server-url flag or from a Kubernetes context using –from-context flag. The default is to list accelerators from the Kubernetes context. To override this, you can set the ACC_SERVER_URL environment variable with the URL for the Application Accelerator server you want to access.

```
tanzu accelerator list [flags]
```

Examples

```
tanzu accelerator list
```

Options

```
      --from-context        retrieve resources from the current context defined in kub
econfig
  -h, --help                help for list
  -n, --namespace string    namespace for accelerator system (default "accelerator-sys
tem")
      --server-url string   the URL for the Application Accelerator server
  -t, --tags strings        accelerator tags to match against
  -v, --verbose             include repository and show long URLs or image digests in
the output
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

tanzu accelerator

tanzu accelerator push

tanzu accelerator push

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator push` command to push source code from local path to source image.

Synopsis

Push source code from local path to source image used by an accelerator

```
tanzu accelerator push [flags]
```

Examples

```
tanzu accelerator push --local-path <local path> --source-image <image>
```

Options

```
  -h, --help                  help for push
      --local-path string     path to the directory containing the source for the acce
lerator
      --source-image string   name of the source image for the accelerator
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

tanzu accelerator update

This topic tells you how to use the Tanzu Accelerator CLI `tanzu accelerator update` command to update an accelerator.

Synopsis

Update an accelerator resource with the specified name using the specified configuration.

Accelerator configuration options include: - Git repository URL and branch/tag where accelerator code and metadata is defined - Metadata like description, display-name, tags and icon-url

The update command also provides a –reoncile flag that will force the accelerator to be refreshed with any changes made to the associated Git repository.

```
tanzu accelerator update [flags]
```

Examples

```
tanzu accelerator update <accelerator-name> --description "Lorem Ipsum"
```

Options

```
      --description string    description of this accelerator
      --display-name string   display name for the accelerator
      --git-branch string     Git repository branch to be used
      --git-repo string       Git repository URL for the accelerator
      --git-sub-path string   Git repository subPath to be used
      --git-tag string        Git repository tag to be used
  -h, --help                  help for update
      --icon-url string       URL for icon to use with the accelerator
```

```
      --interval string       interval for checking for updates to Git or image reposi
tory
  -n, --namespace string      namespace for accelerator system (default "accelerator-s
ystem")
      --reconcile             trigger a reconciliation including the associated GitRep
ository resource
      --secret-ref string     name of secret containing credentials for private Git or
image repository
      --source-image string   name of the source image for the accelerator
      --tags strings          tags that can be used to search for accelerators
```

Options inherited from parent commands

```
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)
```

SEE ALSO

- tanzu accelerator - Manage accelerators in a Kubernetes cluster

Overview of the Tanzu Insight plug-in

The Tanzu Insight CLI plug-in helps you query vulnerability, image, and package data.

Follow these steps to install, configure, and use your Tanzu Insight CLI plug-in:

Note: Prior to using the CLI plug-in, you must install the Supply Chain Security Tools - Store, either as its own package, or as part of Tanzu Application Platform View profile.

1. If the `insight` plug-in is not already installed, see Install the Tanzu Insight plug-in
2. Configure insight

Once `tanzu insight` CLI plug-in is set up:

1. Add data
2. Query data

Overview of the Tanzu Insight plug-in

The Tanzu Insight CLI plug-in helps you query vulnerability, image, and package data.

Follow these steps to install, configure, and use your Tanzu Insight CLI plug-in:

Note: Prior to using the CLI plug-in, you must install the Supply Chain Security Tools - Store, either as its own package, or as part of Tanzu Application Platform View profile.

1. If the `insight` plug-in is not already installed, see Install the Tanzu Insight plug-in
2. Configure insight

Once `tanzu insight` CLI plug-in is set up:

1. Add data
2. Query data

Install your Tanzu Insight CLI plug-in

This topic tells you how to install your Tanzu Insight CLI plug-in.

Note

Follow the steps in this topic if you do not want to use a profile to install the Tanzu Insight CLI plug-in. For more information about profiles, see About Tanzu Application Platform > components and profiles.

1. From your `tanzu` directory, install the local version of the Tanzu Insight plug-in you downloaded by running:

```
cd $HOME/tanzu
tanzu plugin install insight
```

2. Follow the steps in Configure the Tanzu Insight CLI plug-in.

Configure your Tanzu Insight CLI plug-in

This topic tells you how to configure your Tanzu Insight CLI plug-in.

Set the target and certificate authority (CA) certificate

These instructions are for the recommended configuration where Ingress is enabled. For instructions on non Ingress setups, see Configure target endpoint and certificate.

Set the endpoint host to `metadata-store.INGRESS-DOMAIN`, such as `metadata-store.example.domain.com`. Where `INGRESS-DOMAIN` isthe value of the `ingress_domain` property in your deployment yaml.

Note In a multi-cluster setup, a DNS record is **required** for the domain. The below instructions for single cluster setup do not apply, skip to Set Target section.

Single Cluster setup

In a single-cluster setup, a DNS record is still recommended. However, if no accessible DNS record exists for the domain, edit the `/etc/hosts` file to add a local record:

```
ENVOY_IP=$(kubectl get svc envoy -n tanzu-system-ingress -o jsonpath="{.status.loadBal
ancer.ingress[0].ip}")

# Replace with your domain
METADATA_STORE_DOMAIN="metadata-store.example.domain.com"

# Delete any previously added entry
sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "$ENVOY_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

Set Target

To get the certificate, run:

```
kubectl get secret tap-ingress-selfsigned-root-ca -n cert-manager -o json | jq -r '.da
ta."ca.crt"' | base64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN --ca-cert insight-ca.cr
t
```

Important

The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to add data and query data.

Set the access token

When using the `insight` plug-in, you must set the `METADATA_STORE_ACCESS_TOKEN` environment variable, or use the `--access-token` flag. VMware discourages using the `--access-token` flag as the token appears in your shell history.

The following command retrieves the access token from the default `metadata-store-read-write-client` service account and stores it in `METADATA_STORE_ACCESS_TOKEN`:

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets metadata-store-read-write-cli
ent -n metadata-store -o jsonpath="{.data.token}" | base64 -d)
```

Verify the connection

Verify that your configuration is correct and you can make a connection using `tanzu insight health`.

Important

The `tanzu insight health` command tests the configured endpoint and CA certificate. However, it does not test whether the access token is correct. For that, you must use the plug-in to add and query data.

For example:

```
$ tanzu insight health
Success: Reached Metadata Store!
```

Query vulnerabilities, images, and packages

This topic tells you how to query the database to understand vulnerability, image, and dependency relationships. The Tanzu Insight CLI plug-in queries the database for vulnerability scan reports or Software Bill of Materials (commonly known as SBoM) files.

Supported use cases

The following use cases are supported by the Tanzu Insight CLI plug-in:

- What packages and CVEs exist in a particular image? (`image`)
- What dependencies are affected by a specific CVE? (`vulnerabilities`)

Query using the Tanzu Insight CLI plug-in

There are four commands for querying and adding data:

- `image` - Post an image SBOM or query images for packages and vulnerabilities.
- `package` - Query packages for vulnerabilities or by image or source code.
- `source` - Post a source code SBOM or query source code for packages and vulnerabilities.
- `vulnerabilities` - Query vulnerabilities by image, package, or source code.

For more information about these commands, use `tanzu insight -h` or see Tanzu Insight Details.

Example 1: What packages and CVEs does a specific image contain?

To query an image scan for vulnerabilities, you need the image digest value. Get the image digest value from the image scan resource using Supply Chain Tools - Scan 2.0 or Supply Chain Tools Scan Pre-2.0.

Find the image digest using Supply Chain Tools - Scan 2.0

Find the image digest by looking inside the corresponding image vulnerability scan custom resource.

To get a list of image vulnerability scans, run:

```
kubectl get imagevulnerabilityscan -n WORKLOAD-NAMESPACE
```

For example:

```
$ kubectl get imagevulnerabilityscan -n my-apps
NAME                                  SUCCEEDED   REASON
tanzu-java-web-app-grype-scan-jb76m   True        Succeeded
```

The name of the image vulnerability scan starts with the name of the workload.

To describe the image vulnerability scan, run:

```
kubectl describe imagevulnerabilityscan IMAGE-VULNERABILITY-SCAN-NAME -n WORKLOAD-NAME
SPACE
```

For example:

```
kubectl describe imagevulnerabilityscan tanzu-java-web-app-grype-scan-jb76m -n my-apps
```

In the resource, look for the `Spec.Image` field. The value points to the image that was scanned, including its digest.

For example:

```
Spec:
  Image: fake.oci-registry.io/dev-cluster/supply-chain-apps/tanzu-java-web-app-my-apps
@sha256:a24a8d8eb724b6816f244925cc6625a84c15f6ced6a19335121343424be693cd
```

In this example, the image digest is:
`sha256:a24a8d8eb724b6816f244925cc6625a84c15f6ced6a19335121343424be693cd`

Find the image digest using Supply Chain Tools - Scan Pre-2.0

Find the image digest by looking inside the corresponding image scan custom resource.

Run:

```
kubectl get imagescan WORKLOAD-NAME -n WORKLOAD-NAMESPACE
```

For example:

```
kubectl get imagescan tanzu-java-web-app -n my-apps
```

In the resource, look for the `Spec.Registry.Image` field. The value points to the image that was scanned, including its digest.

For example:

```
Spec:
  Registry:
    Image: fake.oci-registry.io/dev-cluster/supply-chain-apps/tanzu-java-web-app-my-ap
ps@sha256:e8c648533c4c7440ee9a93142ac7480205e0f7669e4f86771cede8bfaacdc2cf
```

In this example, the image digest is:
`sha256:e8c648533c4c7440ee9a93142ac7480205e0f7669e4f86771cede8bfaacdc2cf`

Query an image using the image digest value

When you have found the image digest value, you can query an image using this value.

Run:

```
tanzu insight image get --digest DIGEST
```

Where:

- `DIGEST` is the component version or image digest.

For example:

```
$ tanzu insight image get --digest sha256:sha256:e8c648533c4c7440ee9a93142ac7480205e0f
7669e4f86771cede8bfaacdc2cf
Registry:       fake.oci-registry.com
Image Name:     dev-cluster/supply-chain-apps/tanzu-java-web-app-my-apps
Digest:         sha256:sha256:e8c648533c4c7440ee9a93142ac7480205e0f7669e4f86771cede8bf
aacdc2cf
Packages:
        1. alpine-baselayout@3.1.2-r0
        2. alpine-keys@2.1-r2
        3. apk-tools@2.10.4-r2
        CVEs:
                1. CVE-2021-30139 (High)
                2. CVE-2021-36159 (Critical)
        4. busybox@1.30.1-r3
        CVEs:
                1. CVE-2021-28831 (High)
...
```

Example 2: What packages and CVEs does my source code contain?

When you find the source code organization, repository or commit SHA, you can use these to query the source code in more detail.

Find the source code organization, repository, and commit SHA

To query a source scan for vulnerabilities, you need a Git organization and Git repository, or the commit SHA. Find these by examining the source scan resource.

Run:

```
kubectl describe sourcescan WORKLOAD-NAME -n WORKLOAD-NAMESPACE
```

For example:

```
kubectl describe sourcescan tanzu-java-web-app -n my-apps
```

In the resource look for the `Spec.Blob` field. Within, there's `Revision` and `URL`.

For example:

```
Spec:
  Blob:
    Revision:     master/c7e4c27ba43250a4b7c46f030355c108aa73cc39
    URL:          http://source-controller.flux-system.svc.cluster.local./gitrepositor
y/my-apps/tanzu-java-web-app-gitops/c7e4c27ba43250a4b7c46f030355c108aa73cc39.tar.gz
```

The URL is parsed and split into the organization and repository. Revision is parsed as the commit SHA.

- Organization is parsed as `gitrepository`
- Repository is parsed as `my-apps/tanzu-java-web-app-gitops/c7e4c27ba43250a4b7c46f030355c108aa73cc39.tar.gz`
- Commit SHA is parsed as `master/c7e4c27ba43250a4b7c46f030355c108aa73cc39`

Query the source code using the repository and organization values

Run:

```
tanzu insight source get --repo REPO --org ORG
```

Where:

- `REPO` specifies the repository. For example, `java-web-app`, `my-apps/java-web-app/c7ls8bakd87sakjda8d7.tar.gz`
- `ORG` is the source code's organization. For example, `gitrepository`, `gitrepositiory-kj32kal8`

For example:

```
$ tanzu insight source get --repo my-apps/java-web-app/c7ls8bakd87sakjda8d7.tar.gz --o
rg gitrepository
ID:             1
Repository:  my-apps/java-web-app/c7ls8bakd87sakjda8d7.tar.gz
Commit:  c7e4c27ba43250a4b7c46f030355c108aa73cc39
Organization:   gitrepository
Packages:
                1. go.uber.org/atomic@v1.7.0
                CVEs:
                        1. CVE-2022-42322 (Low)
                2. golang.org/x/crypto@v0.0.0-20220518034528-6f7dac969898
                3. github.com/valyala/bytebufferpool@v1.0.0
```

Query the source code using the commit SHA value

Run:

```
tanzu insight source get --commit COMMIT
```

Where:

- `COMMIT` specifies the commit. For example, d7e4c27ba43250a4b7c46f030355c108aa73cc39, main/d7e4c27ba43250a4b7c46f030355c108aa73cc39

For example:

```
$ tanzu insight source get --commit b66668e
ID:             2
Repository:  kpack
Commit:  b66668e
Organization:   pivotal
Packages:
                1. cloud.google.com/go/kms@v1.0.0
                2. github.com/BurntSushi/toml@v3.1.1
                CVEs:
                        1. CVE-2021-30999 (Low)
                3. github.com/Microsoft/go-winio@v0.5.2
```

Example 3: What dependencies are affected by a specific CVE?

Run:

```
tanzu insight vulnerabilities get --cveid CVE-IDENTIFIER
```

Where:

- `CVE-IDENTIFIER` is the CVE identifier, for example, CVE-2021-30139.

For example:

```
$ tanzu insight vulnerabilities get --cveid CVE-2010-4051
1. CVE-2010-4051 (Low)
Packages:
        1. libc-bin@2.28-10
        2. libc-l10n@2.28-10
        3. libc6@2.28-10
        4. locales@2.28-10
```

Add data

For information about manually adding data, see Add data to your Supply Chain Security Tools - Store.

Add data to your Supply Chain Security Tools - Store

This topic tells you how to add vulnerability scan reports or Software Bill of Materials (commonly known as SBoM) files to your Supply Chain Security Tools (commonly known as SCST) - Store.

Supported formats and file types

Currently, only CycloneDX XML and JSON files are accepted.

Source commits and image files have been tested. Additional file types might work, but are not fully supported (for example, JAR files).

If you are not using a source commit or image file, you must ensure the `component.version` field in the CycloneDX file is non-null.

Generate a CycloneDX file

A CycloneDX file is needed to post data. Supply Chain Security Tools - Scan outputs CycloneDX files automatically. For more information, see Supply Chain Security Tools - Scan.

To generate a file to post manually, use Grype or another tool in the CycloneDX Tool Center.

To use Grype to scan an image and generate an image report in CycloneDX format:

1. Install Grype.
2. Scan the image and generate a report by running:

   ```
   grype REPO:TAG -o cyclonedx > IMAGE-CVE-REPORT
   ```

   Where:

   - `REPO` is the name of your repository
   - `TAG` is the name of a tag
   - `IMAGE-CVE-REPORT` is the resulting file name of the Grype image scan report

   For example:

   ```
   $ grype docker.io/checkr/flagr:1.1.12 -o cyclonedx > image-cve-report
    ✔ Vulnerability DB        [updated]
    ✔ Parsed image
    ✔ Cataloged packages      [21 packages]
    ✔ Scanned image           [8 vulnerabilities]
   ```

Add data with the Tanzu Insight plug-in

Use the following commands to add data:

- `image add`
- `source add`

If you are not using a source commit or image file, you can select either option.

Example #1: Add an image report

To use a CycloneDX-formatted image report:

1. Run:

   ```
   tanzu insight image add --cyclonedxtype TYPE --path IMAGE-CVE-REPORT
   ```

   Where:

   - `TYPE` specifies XML or JSON, the two supported file types
   - `IMAGE-CVE-REPORT` is the location of a Cyclone DX formatted file

   For example:

```
$ tanzu insight image add --cyclonedxtype xml --path downloads/image-cve-report
Image report created.
```

Note

The Metadata Store only stores a subset of CycloneDX file data. Support for more data might be added in the future.

Example #2: Add a source report

To use a CycloneDX-formatted source report:

1. Run:

   ```
   tanzu insight source add --cyclonedxtype TYPE --path SOURCE-CVE-REPORT
   ```

   Where:

   - `TYPE` specifies XML or JSON, the two supported file types
   - `SOURCE-CVE-REPORT` is the location of a Cyclone DX formatted file

   For example:

   ```
   $ tanzu insight source add --cyclonedxtype json --path source-cve-report
   Source report created.
   ```

Note

Supply Chain Security Tools - Store only stores a subset of a CycloneDX file's data. Support for more data might be added in the future.

Tanzu insight CLI plug-in command reference

This topic tells you about the Tanzu Insight CLI plug-in.

Synopsis

This CLI plug-in is used to post data and query the Supply Chain Security Tools - Store through its secure REST API. Source commit and image vulnerability reports can be uploaded using CycloneDX format (XML and JSON) and SPDX format (JSON). Source commit, image, package, and vulnerabilities can be queried and outputted in CycloneDX XML, JSON, and human-readable text formats.

Options

```
-h, --help   help for tanzu insight
```

See also

- Tanzu insight config - Config commands
- Tanzu insight health - Checks if endpoint is reachable

- Tanzu insight image - Image commands
- Tanzu insight package - Package commands
- Tanzu insight source - Source commands
- Tanzu insight version - Display Tanzu Insight version
- Tanzu insight vulnerabilities - Vulnerabilities commands

tanzu insight config set-target

tanzu insight config set-target

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight config set-target` command to set your metadata store endpoint.

Synopsis

Set the target endpoint for the metadata store.

```
tanzu insight config set-target <endpoint> [--ca-cert <ca certificate path to verify p
eer against>] [--access-token <kubernetes service account access token>] [flags]
```

Examples

```
tanzu insight config set-target https://localhost:8443 --ca-cert=/tmp/ca.crt --access-
token eyJhbGc...
```

Options

```
      --access-token string   Kubernetes access token. It is recommended to use the En
vironment Variable METADATA_STORE_ACCESS_TOKEN during the API calls, this will overrid
e access token flag. Note: using the access-token flag stores the token on disk, the E
nvironment Variable is retrieved at the time of the API call
      --ca-cert string        trusted ca certificate
  -h, --help                  help for set-target
```

See also

- Tanzu insight config - Config commands

tanzu insight config

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight config` command to get help for the configuration commands.

Options

```
  -h, --help   help for config
```

See also

- Tanzu insight - This CLI is used to post data and make queries to the metadata store.
- Tanzu insight config set-target - Set metadata store endpoint.

tanzu insight health

tanzu insight health

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight health` command to check if an endpoint is reachable.

Synopsis

Checks if endpoint is reachable.

```
tanzu insight health [flags]
```

Examples

```
tanzu insight health
```

Options

```
  -h, --help   help for health
```

See also

- Tanzu insight

tanzu insight image

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight image` command to get help for the image commands.

Options

```
  -h, --help   help for image
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight image add - Add an image report.
- Tanzu insight image get - Get image by digest.
- Tanzu insight image packages - Get image packages.
- Tanzu insight image vulnerabilities - Get image vulnerabilities.

tanzu insight image add

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight image add` command to add an image report.

```
tanzu insight image add [--cyclonedxtype <json|xml>] [--spdxtype json] --path <filepat
h>
```

If report type is not specified, it will be defaulted to `--cyclonedxtype=xml`

Examples

```
tanzu insight image add --cyclonedxtype json --path /path/to/file.json
```

Options

```
      --cyclonedxtype string   cyclonedx file type(xml/json, default: xml)
  -h, --help                   help for add
      --path string            path to file
      --spdxtype string        spdx file type(json)
```

See also

- Tanzu insight image - Image commands

tanzu insight image get

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight image get` command to get an image by digest.

Synopsis

Get image by digest.

```
tanzu insight image get --digest <image-digest> [--format <image-format>] [flags]
```

Examples

```
tanzu insight image get --digest sha256:a86859ac1946065d93df9ecb5cb7060adeeb0288fad610
b1b659907 --format json
```

Options

```
  -d, --digest string   image digest
  -f, --format string   output format (default "text")
  -h, --help            help for get
```

See Also

- Tanzu insight image - Image commands

tanzu insight image packages

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight image packages` command to get the image packages.

Synopsis

Get image packages.

```
tanzu insight image packages [--digest <image-digest>] [--name <name>] [--format <imag
e-format>] [flags]
```

Examples

```
tanzu insight image packages --digest sha256:a86859ac1946065d93df9ecb5cb7060adeeb0288f
ad610b1b659907 --format json
```

Options

```
  -d, --digest string   image digest
  -f, --format string   output format (default "text")
  -h, --help            help for packages
  -n, --name string     image name
```

See also

- Tanzu insight image - Image commands

tanzu insight image vulnerabilities

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight image vulnerabilities` command to get the image vulnerabilities.

```
tanzu insight image vulnerabilities --digest <image-digest> [--format <image-format>]
[flags]
```

Examples

```
tanzu insight image vulnerabilities --digest sha256:a86859ac1946065d93df9ecb5cb7060ade
eb0288fad610b1b659907 --format json
```

Options

```
  -d, --digest string   image digest
  -f, --format string   output format (default "text")
  -h, --help            help for vulnerabilities
```

See also

- Tanzu insight image - Image commands

tanzu insight package

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight package` command to get help for the package commands.

Options

```
  -h, --help   help for package
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight package get - Get package by name, version, and package manager.
- Tanzu insight package images - Get images that contain the given package by name.
- Tanzu insight package sources - Get sources that contain the given package by name.
- Tanzu insight package vulnerabilities - Get package vulnerabilities.

tanzu insight package get

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight package get` command to get the package by name, version, and package manager.

Synopsis

Get package by name, version, and package manager.

```
tanzu insight package get --name <package name> --version <package version> --pkgmngr
Unknown [--format <format>] [flags]
```

Examples

```
tanzu insight package get --name client --version 1.0.0a --pkgmngr Unknown
```

Options

```
  -f, --format string    output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help             help for get
  -n, --name string      name of the package
  -p, --pkgmngr string   Package manager of the dependency. 'Unknown' is currently the
only supported value (default "Unknown")
  -v, --version string   version of the package
```

See also

- Tanzu insight package - Package commands

tanzu insight package images

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight package images` command to get the images that contain the given package by name.

Synopsis

Get images that contain the given package by name.

```
tanzu insight package images --name <package name> [flags]
```

Examples

```
tanzu insight package images --name client
```

Options

```
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for images
  -n, --name string     name of the package
```

See also

- Tanzu insight package - Package commands

tanzu insight package sources

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight package sources` command to get the sources that contain the given package by name.

Synopsis

Get sources that contain the given package by name.

```
tanzu insight package sources --name <package name> [flags]
```

Examples

```
tanzu insight package sources --name client
```

Options

```
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for sources
  -n, --name string     name of the package
```

See also

- Tanzu insight package - Package commands

tanzu insight package vulnerabilities

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight package vulnerabilities` command to get the package vulnerabilities.

Synopsis

Get package vulnerabilities.

```
tanzu insight package vulnerabilities --name <package name> [flags]
```

Examples

```
tanzu insight package vulnerabilities --name client
```

Options

```
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for vulnerabilities
  -n, --name string     name of the package
```

See also

- Tanzu insight package - Package commands

tanzu insight source

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight source` command to get help for the source commands.

Options

```
  -h, --help   help for source
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight source add - Add a source report.
- Tanzu insight source get - Get sources by repository, commit, or organization.
- Tanzu insight source packages - Get source packages.
- Tanzu insight source vulnerabilities - Get source vulnerabilities.

tanzu insight source add

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight source add` command to add a source report.

```
tanzu insight source add [--cyclonedxtype <json|xml>] [--spdxtype json] --path <filepa
th>
```

If report type is not specified, it defaults to `--cyclonedxtype=xml`

Examples

```
tanzu insight source add --cyclonedxtype json --path  /path/to/file.json
```

Options

```
      --cyclonedxtype string   cyclonedx file type (xml/json, default: xml)
  -h, --help                   help for add
```

```
      --path string            path to file
      --spdxtype string        spdx file type (json)
```

See also

- Tanzu insight source - Source commands

tanzu insight source get

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight source get` command to get sources by repository, commit or organization.

Synopsis

Get sources by repository, commit, or organization.

```
tanzu insight source get --repo <repository> --commit <commit-hash> --org <organizatio
n-name> [--format <format>] [flags]
```

Examples

```
tanzu insight source get --repo github.com/org/example --commit b33dfee51 --org compan
y
```

Options

```
  -c, --commit string   commit hash
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for get
  -o, --org string      organization that owns the source
  -r, --repo string     repository name
```

See also

- Tanzu insight source - Source commands

tanzu insight source packages

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight source packages` command to get the source packages.

Synopsis

Get source packages.

```
tanzu insight source packages [--commit <commit-hash>] [--repo <repo-url>] [--format <
format>] [flags]
```

Examples

```
tanzu insight sources packages --commit 0b1b659907 --format json
```

Options

```
  -c, --commit string   commit hash
  -f, --format string   output format (default "text")
  -h, --help            help for packages
  -r, --repo string     source repository url
```

See also

- Tanzu insight source - Source commands

tanzu insight source vulnerabilities

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight source vulnerabilities` command to get the source vulnerabilities.

Synopsis

Get source vulnerabilities. You can specify either commit or repository.

```
tanzu insight source vulnerabilities [--commit <commit-hash>] [--repo <repo-url>] [--f
ormat <format>] [flags]
```

Examples

```
tanzu insight sources vulnerabilities --commit eb55fc13
```

Options

```
  -c, --commit string   commit hash
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for vulnerabilities
  -r, --repo string     source repository url
```

See also

- Tanzu insight source - Source commands

tanzu insight version

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight version` command to display the Tanzu insight version:

```
tanzu insight version [flags]
```

Options

```
  -h, --help   help for version
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.

tanzu insight vulnerabilities

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight vulnerabilities` command to get help for the vulnerabilities commands.

Options

```
  -h, --help   help for vulnerabilities
```

See also

- Tanzu insight - This CLI is used to post data and query the metadata store.
- Tanzu insight vulnerabilities get - Get vulnerability by CVE id.
- Tanzu insight vulnerabilities images - Get images with a given vulnerability.
- Tanzu insight vulnerabilities packages - Get packages with a given vulnerability.
- Tanzu insight vulnerabilities sources - Get sources with a given vulnerability.

tanzu insight vulnerabilities get

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight vulnerabilities get` command to get a vulnerability by CVE ID.

Synopsis

Get vulnerability by CVE id.

```
tanzu insight vulnerabilities get --cveid <cve-id> [--format <format>] [flags]
```

Examples

```
tanzu insight vulnerabilities get --cveid CVE-123123-2021
```

Options

```
  -c, --cveid string    CVE id
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for get
```

See also

- Tanzu insight vulnerabilities - Vulnerabilities commands

tanzu insight vulnerabilities images

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight vulnerabilities images` command to get the images with a given vulnerability.

Synopsis

Get images with a given vulnerability.

```
tanzu insight vulnerabilities images --cveid <cve-id> [--format <format>] [flags]
```

Examples

```
tanzu insight vulnerabilities images --cveid CVE-123123-2021
```

Options

```
  -c, --cveid string    CVE id
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for images
```

See also

- Tanzu insight vulnerabilities - Vulnerabilities commands

tanzu insight vulnerabilities packages

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight vulnerabilities packages` command to get the packages with a given vulnerability.

Synopsis

Get packages with a given vulnerability.

```
tanzu insight vulnerabilities packages --cveid <cve-id> [--format <format>] [flags]
```

Examples

```
tanzu insight vulnerabilities packages --cveid CVE-123123-2021
```

Options

```
  -c, --cveid string    CVE id
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for packages
```

See also

- Tanzu insight vulnerabilities - Vulnerabilities commands

tanzu insight vulnerabilities sources

This topic tells you how to use the Tanzu Insight CLI plug-in `tanzu insight vulnerabilities sources` command to get the sources with a given vulnerability.

Synopsis

Get sources with a given vulnerability.

```
tanzu insight vulnerabilities sources --cveid <cve-id> [--format <format>] [flags]
```

Examples

```
tanzu insight vulnerabilities sources --cveid CVE-123123-2021
```

Options

```
  -c, --cveid string    CVE id
  -f, --format string   output format which can be in 'json' or 'text'. If not presen
t, defaults to text. (default "text")
  -h, --help            help for sources
```

See also

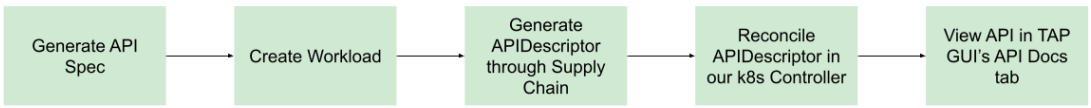
- Tanzu insight vulnerabilities - Vulnerabilities commands

Overview of API Auto Registration

This topic provides an overview of API Auto Registration for Tanzu Application Platform.

Overview

API Auto Registration automates the registration of API specification defined in a workload's configuration. The registered API specification is accessible in Tanzu Application Platform GUI without any additional steps. An automated workflow using a supply chain, leverages API Auto Registration to create and manage a Kubernetes Custom Resource (CR) of kind `APIDescriptor`. A Kubernetes controller periodically reconciles the CR and updates the API entity in Tanzu Application Platform GUI to achieve automated API specification registration from origin workloads. You might also use API Auto Registration without supply chain automation, with other GitOps processes, or by directly applying an `APIDescriptor` CR to the cluster.



Getting started

For information about API Auto Registration architecture, or the APIDescriptor CR and API entities in Tanzu Application Platform GUI, see Key Concepts.

For information about configuring iterate, run, and full Tanzu Application Platform cluster profiles, see Configure API Auto Registration.

For information about generating API specifications and registering them with Tanzu Application Platform GUI catalog, see Use API Auto Registration.

For information about other profiles, install the `api-auto-registration` package. See Install API Auto Registration.

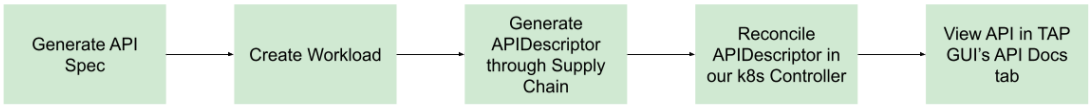
For information about troubleshooting and debugging API Auto Registration, see Troubleshooting.

Overview of API Auto Registration

This topic provides an overview of API Auto Registration for Tanzu Application Platform.

Overview

API Auto Registration automates the registration of API specification defined in a workload's configuration. The registered API specification is accessible in Tanzu Application Platform GUI without any additional steps. An automated workflow using a supply chain, leverages API Auto Registration to create and manage a Kubernetes Custom Resource (CR) of kind `APIDescriptor`. A Kubernetes controller periodically reconciles the CR and updates the API entity in Tanzu Application Platform GUI to achieve automated API specification registration from origin workloads. You might also use API Auto Registration without supply chain automation, with other GitOps processes, or by directly applying an `APIDescriptor` CR to the cluster.



Getting started

For information about API Auto Registration architecture, or the APIDescriptor CR and API entities in Tanzu Application Platform GUI, see Key Concepts.

For information about configuring iterate, run, and full Tanzu Application Platform cluster profiles, see Configure API Auto Registration.

For information about generating API specifications and registering them with Tanzu Application Platform GUI catalog, see Use API Auto Registration.

For information about other profiles, install the `api-auto-registration` package. See Install API Auto Registration.

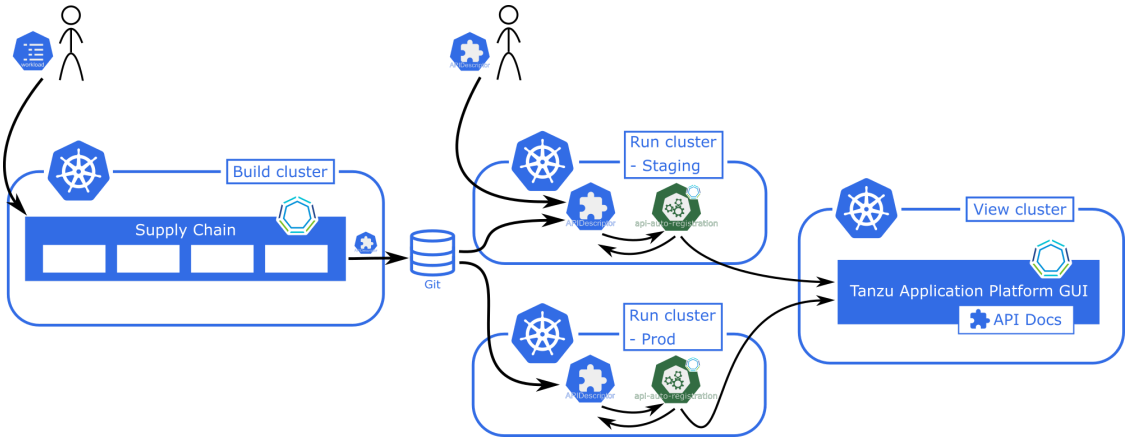
For information about troubleshooting and debugging API Auto Registration, see Troubleshooting.

Key Concepts for API Auto Registration

This topic explains key concepts you use with API Auto Registration.

API Auto Registration Architecture

You can use the full potential of API Auto Registration by using a distributed environment, like the one in this diagram:



APIDescriptor Custom Resource Explained

To use API Auto Registration, you must create a custom resource of type `APIDescriptor`. The information from this custom resource is used to construct an API entity in Tanzu Application Platform GUI.

This custom resource exposes the following text boxes:

```
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
  name:                   # name of your APIDescriptor
  namespace:              # optional namespace of your APIDescriptor
spec:
  type:                   # type of the API spec. oneOf(openapi, grpc, asyncapi, graphq
l)
  description:            # description for the API exposed
  system:                 # system that the API is part of
  owner:                  # person/team that owns the API
  location:
    path:                 # sub-path where the API spec is available
    baseURL:              # base URL object where the API spec is available. oneOf(url,
ref)
      url:                # static absolute base URL
      ref:                # object ref to oneOf(HTTPProxy, Knative Service, Ingress)
        apiVersion:
        kind:
        name:
        namespace:
```

The text boxes cause specific behavior in Tanzu Application Platform GUI:

- The system and owner are copied to the API entity. You might have to separately create and add the System and Group kind to the catalog.
- Tanzu Application Platform GUI uses the namespace for the API entity where the APIDescriptor CR is applied. This causes the API entity's name, system, and owner to all be in that namespace.
- To explicitly use a system or owner in a different namespace, you can specify that in the `system: my-namespace/my-other-system` or `owner: my-namespace/my-other-team` text boxes.
- If the system or owner you are trying to link doesn't have a namespace specified, you can qualify them with the `default` namespace. For example, `system: default/my-default-system`

With an Absolute URL

To create an APIDescriptor with a static `baseURL.url`, you must apply the following YAML to your cluster.

```
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
  name: sample-absolute-url
spec:
  type: openapi
  description: A set of API endpoints to manage the resources within the petclinic ap
p.
  system: spring-petclinic
  owner: team-petclinic
  location:
    path: "/v3/api-docs.yaml"
    baseURL:
      url: https://myservice.com
```

With an Object Ref

You can use an object reference, instead of hard coding the URL, to point to a HTTPProxy, Knative Service, or Ingress.

With an HTTPPRoxy Object Ref

This section includes an example YAML that points to an HTTPProxy from which the controller extracts the `.spec.virtualhost.fqdn` as the baseURL.

```
apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
kind: APIDescriptor
metadata:
  name: sample-contour-ref
spec:
  type: openapi
  description: A set of API endpoints to manage the resources within the petclinic ap
p.
  system: spring-petclinic
  owner: team-petclinic
  location:
    path: "/test/openapi"
    baseURL:
      ref:
        apiVersion: projectcontour.io/v1
        kind: HTTPProxy
        name: my-httpproxy
        namespace: my-namespace # optional
```

With a Knative Service Object Ref

To use a Knative Service, your controller reads the `status.url` as the baseURL. For example:

```
# all other fields similar to the above example
    baseURL:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: my-knative-service
        namespace: my-namespace # optional
```

With an Ingress Object Ref

To use an Ingress instead, your controller reads the URL from the jsonPath specified. When jsonPath is left empty, your controller reads the `"{.spec.rules[0].host}"` as the URL. For example:

```
# all other fields similar to the above example
    baseURL:
      ref:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: my-ingress
        jsonPath: "{.spec.rules[1].host}"
        namespace: my-namespace # optional
```

APIDescriptor Status Fields

When processing an APIDescriptor several fields are added to the `status`. One of these is `conditons`, which provide information useful for troubleshooting. The conditions are explained in the Troubleshooting Guide.

In addition to `conditions` the `status` contains a couple of other useful fields. The following is a list of these fields with a brief explanation of what they contain.

```
status:
  registeredEntityURL:   # Url of the corresponding API Entity in TAP GUI
  registeredTapUID:      # Unique identifier for the corresponding API Entity in TAP G
UI
  resolvedAPISpec:       # Full API Spec as retrieved by Api Auto Registration
```

Install API Auto Registration

This topic describes how you can install API Auto Registration from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install API Auto Registration. For more information about profiles, see Components and installation profiles.

Tanzu Application Platform prerequisites

Before installing API Auto Registration, complete all prerequisites to install Tanzu Application Platform. See Tanzu Application Platform Prerequisites.

Using with TLS

Starting in Tanzu Application Platform v1.4, TLS is turned on by default for several components. API Auto Registration automatically trusts the CA for the shared `ingress_issuer` when using the default ClusterIssuer `tap-ingress-selfsigned`. This change means that a `Certificate` is automatically generated using this issuer.

If you do not want a `Certificate` to generate automatically, you can set the `auto_generate_cert` flag to `false` in the values file. To replace the default with a custom ingress issuer, see Security and compliance. Whenever you do not use the default ClusterIssuer `tap-ingress-selfsigned`, do not

automatically generate certificates, or use other custom CAs, you must manually set the certificate. See Troubleshooting.

Install

To install the API Auto Registration package:

1. List version information for the package by running:

   ```
   tanzu package available list apis.apps.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list apis.apps.tanzu.vmware.com --namespace tap-insta
   ll
   - Retrieving package versions for apis.apps.tanzu.vmware.com...
     NAME                                VERSION  RELEASED-AT
     apis.apps.tanzu.vmware.com  0.1.0        2022-08-30 19:00:00 -0500 -05
     apis.apps.tanzu.vmware.com  0.2.0        2022-11-24 12:20:00 -0500 -05
   ```

2. (Optional) Gather values schema.

   Display values schema of the package:

   ```
   tanzu package available get apis.apps.tanzu.vmware.com/VERSION-NUMBER --values-
   schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package you retrieved.

   For example:

   ```
   tanzu package available get apis.apps.tanzu.vmware.com/0.2.2 --values-schema --
   namespace tap-install

   Retrieving package details for apis.apps.tanzu.vmware.com/0.2.2...
   KEY                       DEFAULT                                          TYPE
   DESCRIPTION
   ca_cert_data                                                               string
   Optional: PEM-encoded certificate data for the controller to trust TLS.
   ingress_issuer                                                             string
   Optional: Name of the default cluster issuer used to generate certificates
   auto_generate_cert        true                                             boolea
   n  Flag that indicates if a cert-manager certificate should be generated using
   the ingress_issuer. Only applies if the ingress_issuer is specified
   connections with a custom CA
   cluster_name              dev                                              string
   Name of the cluster used for setting the API entity lifecycle in TAP GUI. The v
   alue should be unique for each run cluster.
   sync_period               5m                                               string
   Time period used for reconciling an APIDescriptor.
   tap_gui_url               http://server.tap-gui.svc.cluster.local:7000  string
   FQDN URL for TAP GUI.
   replicas                  1                                                intege
   r  Number of controller replicas to deploy.
   resources.limits.cpu      500m                                             string
   CPU limit of the controller.
   resources.limits.memory   500Mi                                            string
   Memory limit of the controller.
   resources.requests.cpu    20m                                              string
   CPU request of the controller.
   resources.requests.memory  100Mi                                           string
   Memory request of the controller.
   logging_profile           production                                       string
   ```

```
Logging profile for controller. If set to development, use console logging with
full stack traces, else use JSON logging.
```

3. Locate the Tanzu Application Platform GUI URL.

   When running on a full profile Tanzu Application Platform cluster, the default value of Tanzu Application Platform GUI URL is sufficient. You can edit this to match the externally available FQDN of Tanzu Application Platform GUI to display the entity URL in the externally accessible APIDescriptor status.

   When installed in a run cluster or with a profile where Tanzu Application Platform GUI is not installed in the same cluster, you must set the `tap_gui_url` parameters correctly for successful entity registration with Tanzu Application Platform GUI.

   You can locate the `tap_gui_url` by going to the view cluster with the Tanzu Application Platform GUI you want to register the entity with:

   ```
   kubectl get secret tap-values -n tap-install -o jsonpath="{.data['tap-values\.y
   aml']}" | base64 -d | yq '.tap_gui.app_config.app.baseUrl'
   ```

4. (Optional) VMware recommends creating `api-auto-registration-values.yaml`.

   To overwrite the default values when installing the package, create a `api-auto-registration-values.yaml` file:

   ```
   tap_gui_url: https://tap-gui.view-cluster.com
   cluster_name: staging-us-east
   ca_cert_data:  |
       -----BEGIN CERTIFICATE-----
       MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
       -----END CERTIFICATE-----
   sync_period: 2m
   ```

5. Install the package using the Tanzu CLI:

   ```
   tanzu package install api-auto-registration
   --package apis.apps.tanzu.vmware.com
   --namespace tap-install
   --version VERSION-NUMBER
   --values-file api-auto-registration-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the package you retrieved in the earlier step.

6. Verify the package installation by running:

   ```
   tanzu package installed get api-auto-registration -n tap-install
   ```

   Verify that `STATUS` is `Reconcile succeeded`:

   ```
   kubectl get pods -n api-auto-registration
   ```

7. Verify that applying an APIDescriptor resource to your cluster causes the `STATUS` showing `Ready`:

   ```
   kubectl apply -f - <<EOF
   apiVersion: apis.apps.tanzu.vmware.com/v1alpha1
   kind: APIDescriptor
   metadata:
     name: sample-api-descriptor-with-absolute-url
   spec:
     type: openapi
     description: A sample APIDescriptor to validate package installation successf
   ```

```
ul
  system: test-installation
  owner: test-installation
  location:
    path: "/api/v3/openapi.json"
    baseURL:
      url: https://petstore3.swagger.io
EOF
```

Verify that the APIDescriptor status shows `Ready`:

```
kubectl get apidescriptors
NAME                                        STATUS
sample-api-descriptor-with-absolute-url     Ready

kubectl get apidescriptors -owide
NAME                                        STATUS     TAP GUI ENTITY URL      API
SPEC URL
sample-api-descriptor-with-absolute-url     Ready      <url-to-the-entity>     <ur
l-to-the-api-spec>
```

If the status does not show `Ready`, you can inspect the reason with the detailed message shown by running:

```
kubectl get apidescriptor sample-api-descriptor-with-absolute-url -o jsonpath
='{.status.conditions[?(@.type=="Ready")].message}'
```

Verify that the entity is created in your Tanzu Application Platform GUI: `TAP-GUI-URL/catalog/default/api/sample-api-descriptor-with-absolute-url`

Use API Auto Registration

This topic describes how you can use API Auto Registration.

Note

The run profile requires you to update the install values before proceeding. For iterate and full profiles, the default values work but you might prefer to update them. For information about profiles, see About Tanzu Application Platform profiles.

API Auto Registration requires the following:

1. A location exposing a dynamic or static API specification.
2. An APIDescriptor Custom Resource (CR) with that location created in the cluster.
3. (Optional) Configure Cross-Origin Resource Sharing (CORS) for OpenAPI specifications.

To generate OpenAPI Spec:

- By creating a simple Spring Boot app
- By scaffolding a new project using App Accelerator Template
- In an existing Spring Boot project

To create APIDescriptor Custom Resource:

- Using Out Of The Box Supply Chains
- Using Custom Supply Chains
- Using other GitOps processes or Manually

To configure:

- CORS for viewing OpenAPI Spec in TAP GUI

Generate OpenAPI Spec

Using a Spring Boot app with a REST service

You can use a Spring Boot example app built using Building a RESTful Web Service guide. and has the Springdoc dependency.

Example of a workload using the Spring Boot app:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: simple-rest-app
  labels:
    ...
    apis.apps.tanzu.vmware.com/register-api: "true"
spec:
  source:
    ...
  params:
    - name: api_descriptor
      value:
        type: openapi
        location:
          path: "/v3/api-docs"
        system: dev
        owner: team-a
        description: "A set of API endpoints."
```

Using App Accelerator Template

If you are creating a new application exposing an API, you might use the java-rest-service App Accelerator template to get a pre-built app that includes a `workload.yaml` with a basic REST API. From your Tanzu Application Platform GUI Accelerators tab, search for the accelerator and scaffold it according to your needs.

Using an existing Spring Boot project using springdoc

If you have an existing Spring Boot app that exposes an API, you can generate OpenAPI specifications using springdoc. See the springdoc documentation

After you have springdoc configured and an OpenAPI automatically generated, you can choose one of the three methods of creating the APIDescriptor custom resource. VMware recommends having your Spring Boot app to be managed using Workloads and the Out-Of-The-Box (OOTB) supply chain. See the Use Out-Of-The-Box (OOTB) supply chains for further instructions. Alternatively, if you want to use custom supply chains, see Using Custom Supply Chains. Lastly, if you want to use a different Gitops process or manage the APIDescriptor CR manually, see the Using other GitOps processes or Manually section.

Create APIDescriptor Custom Resource

Use Out-Of-The-Box (OOTB) supply chains

All the Out-Of-The-Box (OOTB) supply chains are modified so that they can use API Auto Registration. If you want your workload to be auto registered, you must make modifications to your

workload YAML:

1. Add the label `apis.apps.tanzu.vmware.com/register-api: "true"`.
2. Add a parameter of `type api_descriptor`:

```
  params:
    - name: api_descriptor
      value:
        type: openapi   # We currently support any of openapi, aysncapi, graphq
l, grpc
        location:
          path: "/v3/api-docs"  # The path to the api documentation
        owner: team-petclinic   # The team that owns this
        description: "A set of API endpoints to manage the resources within the
petclinic app."
```

There are 2 different options for the location:

- The default supply chains use Knative to deploy your applications. In this event the only location information you must send is the path to the API documentation. The controller can figure out the base URL for you.
- You can hardcode the URL using the baseURL property. The controller uses a combination of this baseURL and your path to retrieve the YAML.

Example workload that exposes a Knative service:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: petclinic-knative
  labels:
    ...
    apis.apps.tanzu.vmware.com/register-api: "true"
spec:
  source:
    ...
  params:
    - name: api_descriptor
      value:
        type: openapi
        location:
          path: "/v3/api-docs"
        system: pet-clinics
        owner: team-petclinic
        description: "A set of API endpoints to manage the resources within the petcli
nic app."
```

Example of a workload with a hardcoded URL to the API documentation:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: petclinic-hard-coded
  labels:
    ...
    apis.apps.tanzu.vmware.com/register-api: "true"
spec:
  source:
    ...
  params:
    - name: api_descriptor
      value:
```

```
        type: openapi
        location:
          baseURL: http://petclinic-hard-coded.my-apps.tapdemo.vmware.com/
          path: "/v3/api-docs"
        owner: team-petclinic
        system: pet-clinics
        description: "A set of API endpoints to manage the resources within the petcli
nic app."
```

After the supply chain runs, it creates an `APIDescriptor` custom resource. This resource is what Tanzu Application Platform uses to auto register your API. See APIDescriptor explained.

Using Custom Supply Chains

If you are creating custom supply chains, you can still use API Auto Registration. To write a supply chain pipeline, use `ClusterConfigTemplate` by the name of `config-template` in your pipeline. To write a custom task, verify how the template is written to read parameters, interpret baseURL from Knative Services, and construct APIDescriptor CRs.

In the Delivery pipeline, you must directly create an APIDescriptor custom resource. You must grant permissions to create the CR from the delivery pipeline.

For information about APIDescriptors, see Key Concepts.

Using other GitOps processes or Manually

Using your GitOps process, or manually, you must stamp out an APIDescriptor CR and apply it in the cluster you choose. Be sure specify all the required fields for an APIDescriptor CR to reconcile.

For information about APIDescriptors, see Key Concepts.

Additional configuration

Setting up CORS for OpenAPI specifications

The agent, usually a browser, uses the CORS protocol to verify whether the current origin uses an API. To use the “Try it out” feature for OpenAPI specifications from the API Documentation plug-in, you must configure CORS to allow successful requests.

Your API must be configured to allow CORS Requests from Tanzu Application Platform GUI. How you accomplish this varies based on the programming language and framework you are using. If you are using Spring, see CORS support in spring framework.

At a high level, the Tanzu Application Platform GUI domain must be accepted as valid cross-origin by your API.

Verify the following:

- **Origins allowed** header: `Access-Control-Allow-Origin`: A list of comma-separated values. This list must include your Tanzu Application Platform GUI host.
- **Methods allowed** header: `Access-Control-Allow-Method`: Must allow the method used by your API. Also confirm that your API supports preflight requests, a valid response to the OPTIONS HTTP method.
- **Headers allowed** header: `Access-Control-Allow-Headers`: If the API requires any header, you must include it in the API configuration or your authorization server.

Troubleshoot API Auto Registration

This topic contains ways that you can troubleshoot API Auto Registration.

Debug API Auto Registration

This section includes commands for debugging or troubleshooting the APIDescriptor CR.

1. Get the details of APIDescriptor CR.

   ```
   kubectl get apidescriptor <api-apidescriptor-name> -owide
   ```

2. Find the status of the APIDescriptor CR.

   ```
   kubectl get apidescriptor <api-apidescriptor-name> -o jsonpath='{.status.condit
   ions}'
   ```

3. Read logs from the `api-auto-registration` controller.

   ```
   kubectl -n api-auto-registration logs deployment.apps/api-auto-registration-con
   troller
   ```

4. Patch an APIDescriptor that is stuck in Deleting mode.

   This might happen if the controller package is uninstalled before you clean up the APIDescriptor resources. You can reinstall the package and delete all the APIDescriptor resources first, or run the following command for each stuck APIDescriptor resource.

   ```
   kubectl patch apidescriptor <api-apidescriptor-name> -p '{"metadata":{"finalize
   rs":null}}' --type=merge
   ```

   Note

   If you manually remove the finalizers from the APIDescriptor resources, you can have stale API entities within Tanzu Application Platform GUI that you must manually deregister.

APIDescriptor CRD shows message of `connection refused` but service is up and running

In Tanzu Application Platform v1.4 and later, if your workloads use ClusterIssuer for the TLS configuration or your API specifications location URL is secured using a custom CA, your APIDescription CRD shows a status and message similar to:

```
    Message:               Get "https://spring-petclinic.example.com/v3/api-docs": dia
l tcp 12.34.56.78:443: connect: connection refused
    Reason:                FailedToRetrieve
    Status:                False
    Type:                  APISpecResolved
    Last Transition Time:  2022-11-28T09:59:13Z
```

This might be due to your workloads using a custom Ingress issuer. To solve this issue, either:

- Configure `ca_cert_data` following the instructions in Configure CA Cert Data.
- Deactivate TLS by setting `shared.ingress_issuer: ""`. VMware discourages this method. Deactivating TLS reduces your ability to test plugin functionality and iterate quickly.

Configure CA Cert Data

1. Obtain the PEM Encoded crt file for your ClusterIssuer or TLS setup . You use this to update the `api-auto-registration` package.

2. If you installed the API Auto Registration package through predefined profiles, you must update the `tap-values.yaml` and update the Tanzu Application Platform installation. Place the PEM encoded certificate into the `shared.ca_cert_data` key of the values file. See Install your Tanzu Application Platform profile. Run the following command to update the package.

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
   lues-file tap-values.yaml -n tap-install
   ```

   Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

3. If you installed the API Auto Registration package as standalone, you must update the `api-auto-registration-values.yaml` and then update the package. Place the PEM encoded certificate into the `ca_cert_data` key of the values file. Run to update the package.

   ```
   tanzu package installed update api-auto-registration --version API-AUTO-REGISTR
   ATION-VERSION --namespace tap-install --values-file api-auto-registration-value
   s.yaml
   ```

   Where `API-AUTO-REGISTRATION-VERSION` is the version of API Auto Registration installed.

   You can find the available api-auto-registration versions by running:

   ```
   tanzu package available list -n tap-install | grep 'API Auto Registration'
   ```

APIDescriptor CRD shows message of `x509: certificate signed by unknown authority` but service is running

Your APIDescription CRD shows a status and message similar to:

```
    Message:               Put "https://tap-gui.tap.my-cluster.tapdemo.vmware.com/api/
catalog/immediate/entities": x509: certificate signed by unknown authority
    Reason:                Error
    Status:                False
    Type:                  Ready
    Last Transition Time:  2022-11-28T09:59:13Z
```

This is the same issue as `connection refused` described earlier.

Overview of API portal for VMware Tanzu

You can use API portal for VMware Tanzu to find APIs you can use in your own applications. You can view detailed API documentation and try out an API to meet your needs. API portal assembles its dashboard and detailed API documentation views by ingesting OpenAPI documentation from the source URLs. An API portal operator can add any number of OpenAPI source URLs in a single instance.

Getting started

To install the package without the predefined profiles of Tanzu Application Platform, see Install API portal.

For information about API portal for VMware Tanzu, see API portal for VMware Tanzu.

For information about configuring the package, see Configuring API portal for VMware Tanzu on Kubernetes.

API portal for VMware Tanzu supports:

- Authentication through Single Sign-On (SSO)

- API keys configuration and management
- Secure communication by using TLS

Overview of API portal for VMware Tanzu

You can use API portal for VMware Tanzu to find APIs you can use in your own applications. You can view detailed API documentation and try out an API to meet your needs. API portal assembles its dashboard and detailed API documentation views by ingesting OpenAPI documentation from the source URLs. An API portal operator can add any number of OpenAPI source URLs in a single instance.

Getting started

To install the package without the predefined profiles of Tanzu Application Platform, see Install API portal.

For information about API portal for VMware Tanzu, see API portal for VMware Tanzu.

For information about configuring the package, see Configuring API portal for VMware Tanzu on Kubernetes.

API portal for VMware Tanzu supports:

- Authentication through Single Sign-On (SSO)
- API keys configuration and management
- Secure communication by using TLS

Install API portal for VMware Tanzu

This topic tells you how to install and update Tanzu API portal for VMware Tanzu from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install API portal. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing API portal:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install the API portal package:

1. Confirm what versions of API portal are available to install by running:

   ```
   tanzu package available list -n tap-install api-portal.tanzu.vmware.com
   ```

   For example:

```
$ tanzu package available list api-portal.tanzu.vmware.com --namespace tap-inst
all
- Retrieving package versions for api-portal.tanzu.vmware.com...
  NAME                          VERSION  RELEASED-AT
  api-portal.tanzu.vmware.com  1.0.3    2021-10-13T00:00:00Z
```

2. (Optional) Gather values schema.

   ```
   tanzu package available get api-portal.tanzu.vmware.com/VERSION-NUMBER --values
   -schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the API Portal package listed earlier.

   For example:

   ```
   $ tanzu package available get api-portal.tanzu.vmware.com/1.0.3 --values-schema
   --namespace tap-install

   Retrieving package details for api-portal.tanzu.vmware.com/1.0.3...
   ```

3. (Optional) VMware recommends creating `api-portal-values.yaml`.

   To overwrite the default values when installing the package, create a `api-portal-values.yaml` file by following the values schema.

4. Install API portal by running:

   ```
   tanzu package install api-portal -n tap-install -p api-portal.tanzu.vmware.com
   -v VERSION-NUMBER --values-file api-portal-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the API Portal package listed earlier.

   For example:

   ```
   $ tanzu package install api-portal -n tap-install -p api-portal.tanzu.vmware.co
   m -v 1.0.3 --values-file api-portal-values.yaml

   / Installing package 'api-portal.tanzu.vmware.com'
   | Getting namespace 'api-portal'
   | Getting package metadata for 'api-portal.tanzu.vmware.com'
   | Creating service account 'api-portal-api-portal-sa'
   | Creating cluster admin role 'api-portal-api-portal-cluster-role'
   | Creating cluster role binding 'api-portal-api-portal-cluster-rolebinding'
   / Creating package resource
   - Package install status: Reconciling


   Added installed package 'api-portal' in namespace 'tap-install'
   ```

5. Verify the package installation by running:

   ```
   tanzu package installed get api-portal -n tap-install
   ```

   Verify that `STATUS` is `Reconcile succeeded`:

   ```
   kubectl get pods -n api-portal
   ```

Update the installation values for the `api-portal` package

To update the installation values for the `api-portal` package:

1. To overwrite the default values, create new values, or update the existing values, you need an `api-portal-values.yaml` file. If you do not already have an existing values file, you can extract the existing values by running:

   ```
   tanzu package installed get api-portal -n tap-install --values-file-output api-portal-values.yaml
   ```

   You can view the schema of the package:

   ```
   tanzu package available get apis.apps.tanzu.vmware.com/VERSION-NUMBER --values-schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the API Portal package listed in the earlier step.

   For example:

   ```
   tanzu package available get api-portal.tanzu.vmware.com/1.2.5 --values-schema --namespace tap-install
   ```

2. Update the package by using the Tanzu CLI:

   ```
   tanzu package installed update api-auto-registration
   --package apis.apps.tanzu.vmware.com
   --namespace tap-install
   --version VERSION-NUMBER
   --values-file api-portal-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the API Portal package listed in the earlier step.

3. If you installed the API portal package as part of Tanzu Application Platform, you must update the `tap-values.yaml` and update the installation of Tanzu Application Platform. See Install your Tanzu Application Platform profile.

   ```
   tanzu package installed update tap --package tap.tanzu.vmware.com --version VERSION-NUMBER --values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION-NUMBER` is the version of the API Portal package listed in the earlier step.

Note

You can update API portal as part of upgrading Tanzu Application Platform. See Upgrading Tanzu Application Platform.

Overview of API Validation and Scoring

API Validation and Scoring focuses on scanning and validating an OpenAPI specification. The API specification is generated from the API Auto Registration. After an API is registered, the API specification goes through static scan analysis and is validated. Based on the validation, a scoring is provided to indicate the quality and health of the API specification as it relates to Documentation, OpenAPI best practices, and Security. The Validation Analysis card on the API overview page displays the summary of the scores. To learn more details about the scores, you can go to the detailed view by clicking the **MORE DETAILS** link.

API Validation and Scoring helps you to ensure your APIs are secure and robust, by providing feedback and recommendations early on in the software development life cycle. Based on the feedback and recommendations, you can edit your API specifications, improve the scores and the posture of your APIs and better understand how well the APIs are implemented.

Overview of API Validation and Scoring

API Validation and Scoring focuses on scanning and validating an OpenAPI specification. The API specification is generated from the API Auto Registration. After an API is registered, the API specification goes through static scan analysis and is validated. Based on the validation, a scoring is provided to indicate the quality and health of the API specification as it relates to Documentation, OpenAPI best practices, and Security. The Validation Analysis card on the API overview page displays the summary of the scores. To learn more details about the scores, you can go to the detailed view by clicking the **MORE DETAILS** link.

API Validation and Scoring helps you to ensure your APIs are secure and robust, by providing feedback and recommendations early on in the software development life cycle. Based on the feedback and recommendations, you can edit your API specifications, improve the scores and the posture of your APIs and better understand how well the APIs are implemented.

Install API Validation and Scoring

This topic tells you how to install API Validation and Scoring from the Tanzu Application Platform (commonly known as TAP) package repository.

Prerequisites

Before installing API Validation and Scoring, complete the following prerequisites:

1. Create a Tanzu Network account to download Tanzu Application Platform packages.
2. Provision Kubernetes cluster v1.22, v1.23 or v1.24 on Amazon Elastic Kubernetes Service.

   Note

   The Installation of API scoring and validation package must be done on a new cluster without any existing Tanzu Application Platform installations.

3. Install Tanzu CLI.
4. Install kapp.
5. Install Kubernetes CLI. For more information, see Install Tools in the Kubernetes documentation.
6. Deploy Cluster Essentials

Resource requirements

To deploy API Validation and Scoring package, your cluster must have at least:

- 5 nodes.
- 4 vCPUs available per node.
- 16 GB of RAM available per node.
- 100 GB of disk space available across all nodes.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before attempting installation. API Validation and Scoring depends on

VMware Tanzu Network for continued operation. If you don't relocate the images, VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

To relocate images from the VMware Tanzu Network registry to your registry:

1. Set up environment variables for installation use by running:

   ```
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export IMGPKG_REGISTRY_HOSTNAME_1=MY-REGISTRY
   export IMGPKG_REGISTRY_USERNAME_1=MY-REGISTRY-USER
   export IMGPKG_REGISTRY_PASSWORD_1=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_USERNAME="${IMGPKG_REGISTRY_USERNAME_1}"
   export INSTALL_REGISTRY_PASSWORD="${IMGPKG_REGISTRY_PASSWORD_1}"
   export APIX_VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to `MY-REGISTRY`.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own container registry.
   - `MY-TANZUNET-USERNAME` is the user with access to the images in the VMware Tanzu Network registry `registry.tanzu.vmware.com`
   - `MY-TANZUNET-PASSWORD` is the password for `MY-TANZUNET-USERNAME`.
   - `VERSION-NUMBER` is your API Validation and Scoring package version. For example, `0.2.5`
   - `TARGET-REPOSITORY` is your target repository, a folder/repository on `MY-REGISTRY` that serves as the location for the installation files for API Validation and Scoring.

2. Install the Carvel tool imgpkg CLI.

   To query for the available `imgpkg` CLI versions on VMWare Tanzu Network Registry, run:

   ```
   imgpkg tag list -i registry.tanzu.vmware.com/tanzu-application-platform/apix |
   sort -V
   ```

3. Relocate the images with the `imgpkg` CLI by running:

   ```
   imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/apix:${APIX
   _VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/apix
   ```

Add the API Validation and Scoring package repository

Tanzu CLI packages are available on repositories. Adding the API Validation and Scoring package repository makes the packages available for installation.

Relocate images to a registry is strongly recommended but not required for installation. If you skip this step, you can use the following values to replace the corresponding variables:

- `INSTALL_REGISTRY_HOSTNAME` is `registry.tanzu.vmware.com`
- `INSTALL_REPO` is `tanzu-application-platform`.
- `INSTALL_REGISTRY_USERNAME` and `INSTALL_REGISTRY_PASSWORD` are the credentials for the VMware Tanzu Network registry `registry.tanzu.vmware.com`

- `APIX_VERSION` is your API Validation and Scoring package version. For example, `0.2.5`

To add the API Validation and Scoring package repository to your cluster:

1. Create a namespace called `apix-install` for deploying API Validation and Scoring package by running:

   ```
   kubectl create ns apix-install
   ```

   This namespace keeps the objects grouped together logically.

2. Create a secret for adding the API Validation and Scoring package repository:

   ```
   tanzu secret registry add tap-registry --username ${INSTALL_REGISTRY_USERNAME}
   --password ${INSTALL_REGISTRY_PASSWORD} --server ${INSTALL_REGISTRY_HOSTNAME} -
   -export-to-all-namespaces --yes --namespace apix-install
   ```

3. Add the API Validation and Scoring package repository to the cluster by running:

   ```
   tanzu package repository add apix-repository \
   --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/apix:${APIX_VERSION} \
   --namespace apix-install
   ```

4. Verify the package installation by running:

   ```
   tanzu package available list -n apix-install
   ```

   If the package installed, expect to see the output that resembles the following:

   ```
   NAME                          DISPLAY-NAME     SHORT_DESCRIPTION               L
   ATEST-VERSION
   apix.apps.tanzu.vmware.com    apix             apix.apps.tanzu.vmware.com
   0.2.5
   ```

5. Get the status of the API Validation and Scoring package repository by running:

   ```
   tanzu package repository get apix-repository --namespace apix-install
   ```

   For example:

   ```
   ~ %    tanzu package repository get apix-repository --namespace apix-install
   NAME:          apix-repository
   VERSION:       796582
   REPOSITORY:    projects.registry.vmware.com/mazinger/apix
   TAG:           0.2.5
   STATUS:        Reconcile succeeded
   REASON:
   ```

   Verify the `STATUS` is `Reconcile succeeded`

Install

Follow these steps to install the API Validation and Scoring package:

1. To overwrite the default values when installing the package, create the `apix-values.yaml` file:

   ```
   apix:
    host: "HOST"
    backstage:
   ```

```
  host: "BACKSTAGE-HOST"
  port: "BACKSTAGE-PORT"
```

Where:

- `HOST` is the hostname of the API Validation and Scoring GUI. It can be left empty `""` to use the default value.
- `BACKSTAGE-HOST` is the Tanzu Application Platform GUI or Backstage host that you want to point to. For example, `https://tap-gui.view-cluster.com`
- `BACKSTAGE-PORT` is the Tanzu Application Platform GUI or Backstage port that you want to point to. For example, `443`

2. Install the API Validation and Scoring package using the Tanzu CLI by running:

```
tanzu package install apix -n apix-install -p apix.apps.tanzu.vmware.com -v ${A
PIX_VERSION} -f apix-values.yaml
```

3. Verify that STATUS is `Reconcile succeeded` by running:

```
tanzu package installed get apix -n apix-install
```

If your package successfully reconciled, expect to see the output that resembles the following::

```
NAME:                    apix
PACKAGE-NAME:            apix.apps.tanzu.vmware.com
PACKAGE-VERSION:         0.2.5
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Uninstall

Uninstall the API Validation and Scoring package by running:

```
tanzu package installed delete apix -n apix-install
```

For example:

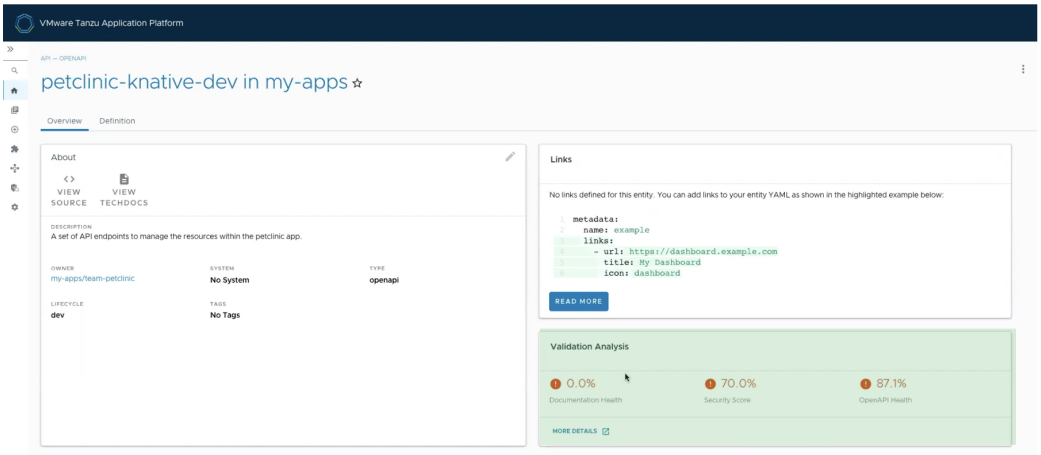
```
% tanzu package installed delete apix -n apix-install
Deleting installed package 'apix' in namespace 'apix-install'. Are you sure? [y/N]: y
Uninstalling package 'apix' from namespace 'apix-install'
Getting package install for 'apix'
Deleting package install 'apix' from namespace 'apix-install'
'PackageInstall' resource deletion status: Deleting
Deleting admin role 'apix-apix-install-cluster-role'
Deleting role binding 'apix-apix-install-cluster-rolebinding'
Deleting secret 'apix-apix-install-values'
Deleting service account 'apix-apix-install-sa'
Uninstalled package 'apix' from namespace 'apix-install'
```

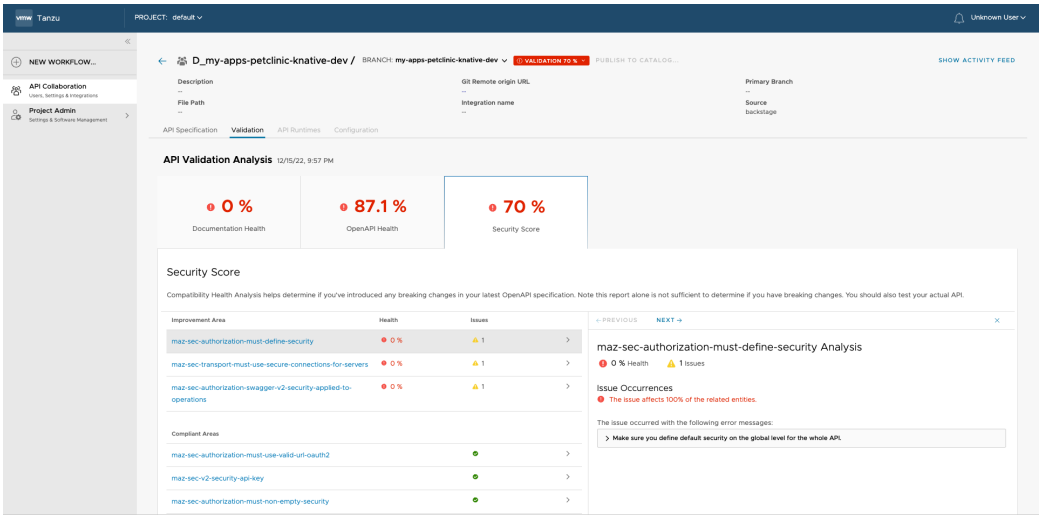
Use API Validation and Scoring to score your auto-registered API

Use API Validation and Scoring to score your auto-registered API

This topic tells you how an Auto Registered API is scored:

- See Use API Auto Registration to deploy the workload.
- Navigate to the Tanzu Application Platform GUI to view the API .
- The **Overview** tab of your API in Tanzu Application Platform GUI shows the API scores.
- To view more details about the Validation Analysis and the required improvements for your API, click **MORE DETAILS**.





Application Accelerator Overview

This topic tells you about the Application Accelerator component and architecture in Tanzu Application Platform (commonly known as TAP).

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing your applications and deploying them in a discoverable and repeatable way.

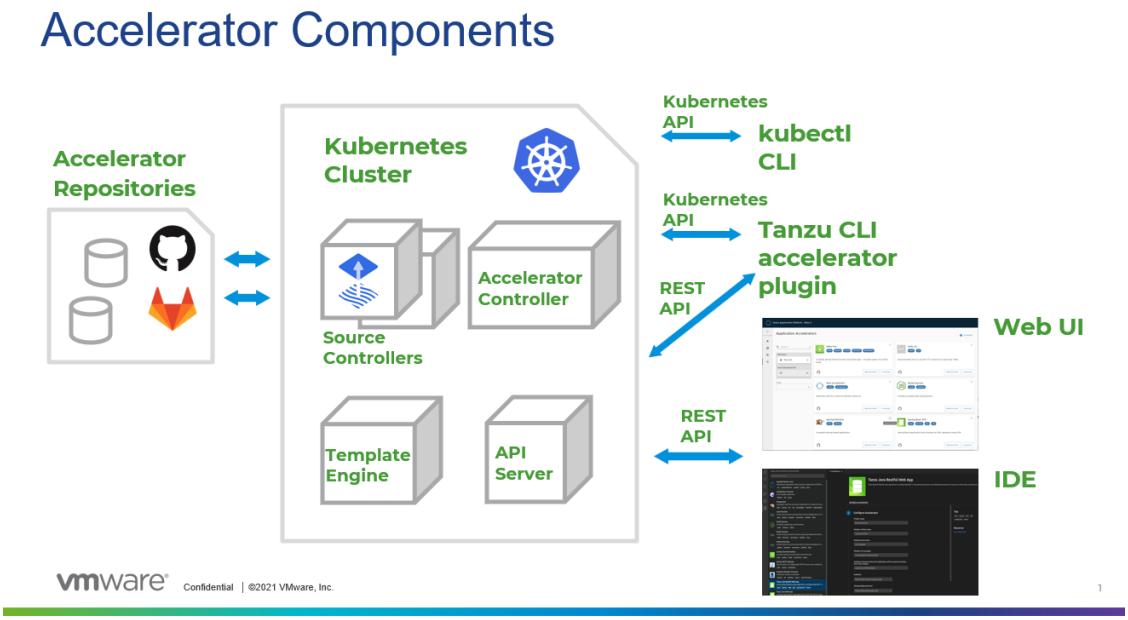
Enterprise Architects author and publish accelerator projects that provide developers and operators in their organization ready-made, conforming code and configurations.

Published accelerator projects are maintained in Git repositories. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

Architecture

The following diagram of Accelerator components illustrates the Application Accelerator architecture.



How does Application Accelerator work?

Application Accelerator allows you to generate new projects from files in Git repositories. An `accelerator.yaml` file in the repository declares input options for the accelerator. This file also contains instructions for processing the files when you generate a new project.

Accelerator custom resources (CRs) control which repositories appear in both the Tanzu Application Platform Application Accelerator UI and in the Application Accelerator extension for VS Code. You can maintain CRs by using Kubernetes tools such as kubectl or by using the Tanzu CLI accelerator commands. The Accelerator controller reconciles the CRs with a Flux2 Source Controller to fetch files from GitHub or GitLab.

The Application Accelerator web UI gives you a searchable list of accelerators to choose from. After you select an accelerator, the UI presents text boxes for the options that are defined within the `accelerator.yaml` of the selected accelerator.

Application Accelerator sends the input values to the Accelerator Engine for processing. (Optional) The user can choose to have a new Git repository created as part of the project creation process. The Engine then returns the project in a ZIP file. If the project was generated using the Application Accelerator extension for VS Code, the project automatically be extracted to the directory location of your choice on your local machine. You can then open the project in your favorite integrated development environment (IDE) to develop further.

Next steps

Learn more about:

- Creating Accelerators

Application Accelerator Overview

This topic tells you about the Application Accelerator component and architecture in Tanzu Application Platform (commonly known as TAP).

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing your applications and deploying them in a discoverable and repeatable way.

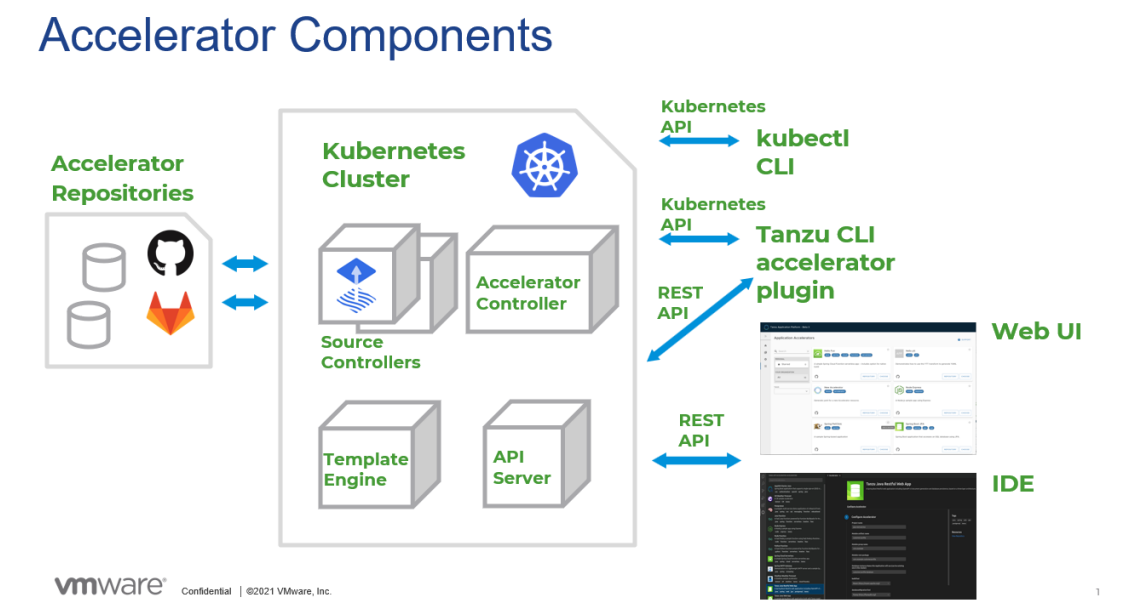
Enterprise Architects author and publish accelerator projects that provide developers and operators in their organization ready-made, conforming code and configurations.

Published accelerator projects are maintained in Git repositories. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

Architecture

The following diagram of Accelerator components illustrates the Application Accelerator architecture.



How does Application Accelerator work?

Application Accelerator allows you to generate new projects from files in Git repositories. An `accelerator.yaml` file in the repository declares input options for the accelerator. This file also contains instructions for processing the files when you generate a new project.

Accelerator custom resources (CRs) control which repositories appear in both the Tanzu Application Platform Application Accelerator UI and in the Application Accelerator extension for VS Code. You can maintain CRs by using Kubernetes tools such as kubectl or by using the Tanzu CLI accelerator commands. The Accelerator controller reconciles the CRs with a Flux2 Source Controller to fetch files from GitHub or GitLab.

The Application Accelerator web UI gives you a searchable list of accelerators to choose from. After you select an accelerator, the UI presents text boxes for the options that are defined within the `accelerator.yaml` of the selected accelerator.

Application Accelerator sends the input values to the Accelerator Engine for processing. (Optional) The user can choose to have a new Git repository created as part of the project creation process. The Engine then returns the project in a ZIP file. If the project was generated using the Application Accelerator extension for VS Code, the project automatically be extracted to the directory location

of your choice on your local machine. You can then open the project in your favorite integrated development environment (IDE) to develop further.

Next steps

Learn more about:

- Creating Accelerators

Install Application Accelerator

This topic tells you how to install Application Accelerator from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Application Accelerator. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before installing Application Accelerator:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Flux SourceController on the cluster. See Install Flux CD Source Controller.
- Install Source Controller on the cluster. See Install Source Controller.

Install

To install Application Accelerator:

1. List version information for the package by running:

   ```
   tanzu package available list accelerator.apps.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list accelerator.apps.tanzu.vmware.com --namespace tap-install

   - Retrieving package versions for accelerator.apps.tanzu.vmware.com...
     NAME                                VERSION  RELEASED-AT
     accelerator.apps.tanzu.vmware.com  1.4.0    2022-12-08 12:00:00 -0500 EST
   ```

2. (Optional) View the changes you can make to the default installation settings by running:

   ```
   tanzu package available get accelerator.apps.tanzu.vmware.com/VERSION-NUMBER \
     --values-schema \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the Application Accelerator package listed earlier.

   For example:

```
$ tanzu package available get accelerator.apps.tanzu.vmware.com/1.4.0 \
    --values-schema \
    --namespace tap-install
```

For more information about values schema options, see the properties listed in Configure properties and resource use later.

3. Create a file named `app-accelerator-values.yaml` using the following example code:

```
server:
  service_type: "LoadBalancer"
  watched_namespace: "accelerator-system"
samples:
  include: true
```

4. Edit the values in your `app-accelerator-values.yaml` if needed, or leave the default values. You can add values you want from Configure properties and resource use.

5. Install the package by running:

```
tanzu package install app-accelerator \
  --package accelerator.apps.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-accelerator-values.yaml
```

Where `VERSION-NUMBER` is the version of the Application Accelerator package listed earlier.

For example:

```
$ tanzu package install app-accelerator \
    --package accelerator.apps.tanzu.vmware.com \
    --version 1.4.0 \
    --namespace tap-install \
    --values-file app-accelerator-values.yaml

- Installing package 'accelerator.apps.tanzu.vmware.com'
| Getting package metadata for 'accelerator.apps.tanzu.vmware.com'
| Creating service account 'app-accelerator-tap-install-sa'
| Creating cluster admin role 'app-accelerator-tap-install-cluster-role'
| Creating cluster role binding 'app-accelerator-tap-install-cluster-rolebindin
g'
| Creating secret 'app-accelerator-tap-install-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'app-accelerator' in namespace 'tap-install'
```

6. Verify the package install by running:

```
tanzu package installed get app-accelerator -n tap-install
```

For example:

```
$ tanzu package installed get app-accelerator -n tap-install

| Retrieving installation details for cc...
NAME:                    app-accelerator
PACKAGE-NAME:            accelerator.apps.tanzu.vmware.com
PACKAGE-VERSION:         1.4.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

7. To see the IP address for the Application Accelerator API when the `server.service_type` is set to `LoadBalancer`, run:

```
kubectl get service -n accelerator-system
```

This lists an external IP address for use with the `--server-url` Tanzu CLI flag for the Accelerator plug-in `generate` & `generate-from-local` command.

For how to troubleshoot installation issues, see Troubleshoot Application Accelerator.

Configure properties and resource use

When you install the Application Accelerator, you can configure the following optional properties from within your `app-accelerator-values.yaml` configuration file:

Property	Default	Description
`registry.secret_ref`	`registry.tanzu.vmware.com`	The secret used for accessing the registry where the App-Accelerator images are located
`server.service_type`	`ClusterIP`	The service type for the acc-ui-server service including LoadBalancer, NodePort, or ClusterIP
`server.watched_namespace`	`accelerator-system`	The namespace the server watches for accelerator resources
`server.engine_invocation_url`	`http://acc-engine.accelerator-system.svc.cluster.local/invocations`	The URL to use for invoking the accelerator engine
`engine.service_type`	`ClusterIP`	The service type for the acc-engine service including LoadBalancer, NodePort, or ClusterIP
`engine.max_direct_memory_size`	`32M`	The maximum size for the Java -XX:MaxDirectMemorySize setting
`samples.include`	`True`	Option to include the bundled sample Accelerators in the installation
`ingress.include`	`False`	Option to include the ingress configuration in the installation
`ingress.enable_tls`	`False`	Option to include TLS for the ingress configuration
`domain`	`tap.example.com`	Top-level domain to use for ingress configuration, default is `shared.ingress_domain`
`tls.secret_name`	`tls`	The name of the secret
`tls.namespace`	`tanzu-system-ingress`	The namespace for the secret
`telemetry.retain_invocation_events_for_no_days`	`30`	The number of days to retain recorded invocation events resources
`telemetry.record_invocation_events`	`true`	The system records each engine invocation when generating files for an accelerator?
`git_credentials.secret_name`	`git-credentials`	The name to use for the secret storing Git credentials for accelerators
`git_credentials.username`	`null`	The user name to use in secret storing Git credentials for accelerators

Property	Default	Description
`git_credentials.password`	`null`	The password to use in secret storing Git credentials for accelerators
`git_credentials.ca_file`	`null`	The CA certificate data to use in secret storing Git credentials for accelerators
`managed_resources.enable`	`false`	Whether to enable the App used to control managed accelerator resources
`managed_resources.git.url`	none	Required if managed_resources are enabled. Git repository URL containing manifests for managed accelerator resources
`managed_resources.git.ref`	`origin/main`	Required if managed_resources are enabled. Git ref to use for repository containing manifests for managed accelerator resources
`managed_resources.git.sub_path`	`null`	Git subPath to use for repository containing manifests for managed accelerator resources
`managed_resources.git.secret_ref`	`git-credentials`	Secret name to use for repository containing manifests for managed accelerator resources

VMware recommends that you do not override the default setting for `registry.secret_ref`, `server.engine_invocation_url`, or `engine.service_type`. These properties are only used to configure non-standard installations.

The following table is the resource use configurations for the components of Application Accelerator.

Component	Resource requests	Resource limits
acc-controller	CPU: 100m memory: 20Mi	CPU: 100m memory: 30Mi
acc-server	CPU: 100m memory:20Mi	CPU: 100m memory: 30Mi
acc-engine	CPU: 500m memory: 1Gi	CPU: 500m memory: 2Gi

Configure Application Accelerator

This topic tells you about advanced configuration options available for Application Accelerator in Tanzu Application Platform (commonly known as TAP). This includes configuring Git-Ops style deployments of accelerators and configurations for use with non-public repositories and in air-gapped environments.

Overview

Accelerators are created either using the Tanzu CLI or by applying a YAML manifest using kubectl. Another option is Using a Git-Ops style configuration for deploying a set of managed accelerators.

Application Accelerator pulls content from accelerator source repositories using either the “Flux SourceController” or the “Tanzu Application Platform Source Controller” components. If the repository used is accessible anonymously from a public server, you do not have to configure anything additional. Otherwise, provide authentication as explained in Using non-public repositories. There are also options for making these configurations easier explained in Configuring tap-values.yaml with Git credentials secret

Using a Git-Ops style configuration for deploying a set of managed accelerators

To enable a Git-Ops style of managing resources used for deploying accelerators, there is a new set of properties for the Application Accelerator configuration. The resources are managed using a Carvel kapp-controller App in the `accelerator-system` namespace that watches a Git repository containing the manifests for the accelerators. This means that you can make changes to the manifests, or to the accelerators they point to, and the changes are reconciled and reflected in the deployed resources.

You can specify the following accelerator configuration properties when installing the Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
accelerator:
  managed_resources:
    enable: true
    git:
      url: GIT-REPO-URL
      ref: origin/main
      sub_path: null
      secret_ref: git-credentials
```

Where:

- `GIT-REPO-URL` is the URL of a Git repository that contains manifest YAML files for the accelerators that you want to have managed. The URL must start with `https://` or `git@`. You can specify a `sub_path` if necessary and also a `secret_ref` if the repository requires authentication. If not needed, then leave these additional properties out.

  For more information, see Configure tap-values.yaml with Git credentials secret and Creating a manifest with multiple accelerators and fragments in this topic.

Functional and Organizational Considerations

Any accelerator manifest that is defined under the `GIT-REPO-URL` and optional `sub_path` is selected by the kapp-controller app. If there are multiple manifests at the defined `GIT-REPO-URL`, they are all watched for changes and displayed to the user as a merged catalog.

For example: if you have two manifests containing multiple accelerator or fragment definitions, `manifest-1.yaml`, and `manifest-2.yaml`, on the same path in the organizational considerations. The resulting catalog is (`manifest-1.yaml` + `manifest-2.yaml`).

Examples for creating accelerators

A minimal example for creating an accelerator

A minimal example might look like the following manifest:

`spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
```

```
    ref:
      branch: main
```

This example creates an accelerator named `spring-cloud-serverless`. The `displayName`, `description`, `iconUrl`, and `tags` text boxes are populated based on the content under the `accelerator` key in the `accelerator.yaml` file found in the `main` branch of the Git repository at Application Accelerator Samples under the sub-path `spring-cloud-serverless`. For example:

`accelerator.yaml`

```
accelerator:
  displayName: Spring Cloud Serverless
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.p
ng
  tags:
  - java
  - spring
  - cloud
  - function
  - serverless
  - tanzu
...
```

To create this accelerator with kubectl, run:

```
kubectl apply --namespace --accelerator-system --filename spring-cloud-serverless.yaml
```

Or, you can use the Tanzu CLI and run:

```
tanzu accelerator create spring-cloud-serverless --git-repo https://github.com/vmware-
tanzu/application-accelerator-samples.git --git-branch main --git-sub-path spring-clou
d-serverless
```

An example for creating an accelerator with customized properties

You can specify the `displayName`, `description`, `iconUrl`, and `tags` text boxes and this overrides any values provided in the accelerator's Git repository. The following example explicitly sets those text boxes and the `ignore` text box:

`my-spring-cloud-serverless.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: my-spring-cloud-serverless
spec:
  displayName: My Spring Cloud Serverless
  description: My own Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.p
ng
  tags:
    - spring
    - cloud
    - function
    - serverless
  git:
    ignore: ".git/, bin/"
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: test
```

To create this accelerator with kubectl, run:

```
kubectl apply --namespace --accelerator-system --filename my-spring-cloud-serverless.y
aml
```

To use the Tanzu CLI, run:

```
tanzu accelerator create my-spring-cloud-serverless --git-repo https://github.com/vmwa
re-tanzu/application-accelerator-samples --git-branch main --git-sub-path spring-cloud
-serverless \
  --description "My own Spring Cloud Function serverless app" \
  --display-name "My Spring Cloud Serverless" \
  --icon-url https://raw.githubusercontent.com/simple-starters/icons/master/icon-clou
d.png \
  --tags "spring,cloud,function,serverless"
```

Note

It is not possible to provide the `git.ignore` option with the Tanzu CLI.

Creating a manifest with multiple accelerators and fragments

You might have a manifest that contains multiple accelerators or fragments. For example:

`accelerator-collection.yaml`

```
---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: spring-cloud-serverless
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples
    subPath: spring-cloud-serverless
    ref:
      branch: main
---
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: tanzu-java-web-app
spec:
  git:
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: tanzu-java-web-app
    ref:
      branch: main
```

For a larger example of this, see Sample Accelerators Main. Optionally, use this to create an initial catalog of accelerators and fragments during a fresh Application Accelerator install.

Configure `tap-values.yaml` with Git credentials secret

Note

For how to create a new OAuth Token for optional Git repository creation, see Create an Application Accelerator Git repository during project creation.

When deploying accelerators using Git repositories that requires authentication or are installed with custom CA certificates, you must provide some additional authentication values in a secret. The examples in the next section provide more details. This section describes how to configure a Git credentials secret that is used in later Git-based examples.

You can specify the following accelerator configuration properties when installing Application Accelerator. The same properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
accelerator:
  git_credentials:
    secret_name: git-credentials
    username: GIT-USER-NAME
    password: GIT-CREDENTIALS
    ca_file: CUSTOM-CA-CERT
```

Where:

- `GIT-USER-NAME` is the user name for authenticating with the Git repository.
- `GIT-CREDENTIALS` is the password or access token used for authenticating with the Git repository. VMware recommends using an access token for this.
- `CUSTOM-CA-CERT` is the certificate data needed when accessing the Git repository.

This is an example of this part of a `tap-values.yaml` configuration:

```
accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret
    ca_file: |
      -----BEGIN CERTIFICATE-----
      .
      .
      .  < certificate data >
      .
      .
      -----END CERTIFICATE-----
```

You can specify the custom CA certificate data using the shared config value `shared.ca_cert_data` and it propagates to all components that can make use of it, including the App Accelerator configuration. The example earlier produces an output such as this using the shared value:

```
shared:
  ca_cert_data: |
    -----BEGIN CERTIFICATE-----
    .
    .
    .  < certificate data >
    .
    .
    -----END CERTIFICATE-----

accelerator:
  git_credentials:
    secret_name: git-credentials
    username: testuser
    password: s3cret
```

Using non-public repositories

For GitHub repositories that aren't accessible anonymously, you must provide credentials in a Secret.

- For HTTPS repositories the secret must contain user name and password fields. The password field can contain a personal access token instead of an actual password. For more information, see Fluxcd/source-controller basic access authentication.
- For HTTPS with self-signed certificates, you can add a `.data.caFile` value to the secret created for HTTPS authentication. For more information, see fluxcd/source-controller HTTPS Certificate Authority.
- For SSH repositories, the secret must contain identity, identity.pub, and known_hosts text boxes. For more information, see fluxcd/source-controller SSH authentication.
- For Image repositories that aren't publicly available, an image pull secret might be provided. For more information, see Kubernetes documentation on using imagePullSecrets.

Examples for a private Git repository

Example using http credentials

To create an accelerator using a private Git repository, first create a secret with the HTTP credentials.

Note

For better security, use an access token as the password.

```
kubectl create secret generic https-credentials \
    --namespace accelerator-system \
    --from-literal=username=USER \
    --from-literal=password=ACCESS-TOKEN
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you created and verified the secret, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
```

```
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: https-credentials
```

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Example using http credentials with self-signed certificate

To create an accelerator using a private Git repository with a self-signed certificate, create a secret with the HTTP credentials and the certificate.

Note

For better security, use an access token as the password.

```
kubectl create secret generic https-ca-credentials \
    --namespace accelerator-system \
    --from-literal=username=USER \
    --from-literal=password=ACCESS-TOKEN \
    --from-file=caFile=PATH-TO-CA-FILE
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system https-ca-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: https-ca-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
  caFile: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
```

```
    secretRef:
      name: https-ca-credentials
```

Important

For https credentials, the `REPOSITORY-URL` must use `https://` as the URL scheme.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Example using SSH credentials

To create an accelerator using a private Git repository, create a secret with the SSH credentials such as this example:

```
ssh-keygen -q -N "" -f ./identity
ssh-keyscan github.com > ./known_hosts
kubectl create secret generic ssh-credentials \
    --namespace accelerator-system \
    --from-file=./identity \
    --from-file=./identity.pub \
    --from-file=./known_hosts
```

If you have a key file already created, skip the `ssh-keygen` and `ssh-keyscan` steps and replace the values for the `kubectl create secret` command. Such as:

- `--from-file=identity=PATH-TO-YOUR-IDENTITY-FILE`
- `--from-file=identity.pub=PATH-TO-YOUR-IDENTITY.PUB-FILE`
- `--from-file=known_hosts=PATH-TO-YOUR-KNOWN-HOSTS-FILE`

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system ssh-credentials -o yaml
```

The output is similar to :

```
apiVersion: v1
kind: Secret
metadata:
  name: ssh-credentials
  namespace: accelerator-system
type: Opaque
data:
  identity: <BASE64>
  identity.pub: <BASE64>
  known_hosts: <BASE64>
```

To use this secret when creating an accelerator, provide the secret name in the `spec.git.secretRef.name` property:

`private-acc-ssh.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
```

```
  git:
    url: REPOSITORY-URL
    ref:
      branch: main
    secretRef:
      name: ssh-credentials
```

When using SSH credentials, the `REPOSITORY-URL` must include the user name as part of the URL. For example: `ssh://user@example.com:22/repository.git`. For more information, see Flux documentation.

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Examples for a private source-image repository

If your registry uses a self-signed certificate then you must add the CA certificate data to the configuration for the “Tanzu Application Platform Source Controller” component. Add it under `source_controller.ca_cert_data` in your `tap-values.yaml` file that is used during installation.

`tap-values.yaml`

```
source_controller:
  ca_cert_data: |-
    -----BEGIN CERTIFICATE-----
    .
    .
    .  < certificate data >
    .
    .
    -----END CERTIFICATE-----
```

Example using image-pull credentials

To create an accelerator using a private source-image repository, create a secret with the image-pull credentials:

```
create secret generic registry-credentials \
    --namespace accelerator-system \
    --from-literal=username=USER \
    --from-literal=password=PASSWORD
```

Verify that your secret was created by running:

```
kubectl get secret --namespace accelerator-system registry-credentials -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  name: registry-credentials
  namespace: accelerator-system
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
```

After you have the secret created, you can create the accelerator by using the `spec.git.secretRef.name` property:

`private-acc.yaml`

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: private-acc
spec:
  displayName: private
  description: Accelerator using a private repository
  source:
    image: "registry.example.com/test/private-acc-src:latest"
    imagePullSecrets:
    - name: registry-credentials
```

If you are using the Tanzu CLI, add the `--secret-ref` flag to your `tanzu accelerator create` command and provide the name of the secret for that flag.

Configure ingress timeouts when some accelerators take longer to generate

If Tanzu Application Platform is configured to use an ingress for Tanzu Application Platform GUI and the Accelerator Server, then it might detect a timeout during accelerator generation. This can happen if the accelerator takes a longer time to generate than the default timeout. When this happens, Tanzu Application Platform GUI appears to continue to run for an indefinite period. In the IDE extension, it shows a `504` error. To mitigate this, you can increase the timeout value for the HTTPProxy resources used for the ingress by applying secrets with overlays to edit the HTTPProxy resources.

Configure an ingress timeout overlay secret for each HTTPProxy

For Tanzu Application Platform GUI, create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-tap-gui-timeout
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "tap-
gui"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "server"}]})
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

For Accelerator Server (used for IDE extension), create the following overlay secret in the `tap-install` namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: patch-accelerator-timeout
  namespace: tap-install
```

```
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"kind": "HTTPProxy", "metadata": {"name": "acce
lerator"}})
    ---
    spec:
      routes:
        #@overlay/match by=overlay.subset({"services": [{"name": "acc-server"}]})
        #@overlay/match-child-defaults missing_ok=True
        - timeoutPolicy:
            idle: 30s
            response: 30s
```

Apply the timeout overlay secrets in tap-values.yaml

Add the following `package_overlays` section to `tap-values.yaml` before installing or updating Tanzu Application Platform:

```
package_overlays:
- name: tap-gui
  secrets:
  - name: patch-tap-gui-timeout
- name: accelerator
  secrets:
  - name: patch-accelerator-timeout
```

Configuring skipping TLS verification for access to Source Controller

You can configure the Flux or Tanzu Application Platform Source Controller to use Transport Layer Security (TLS) and use custom certificates. In that case, configure the Accelerator System to skip the TLS verification for calls to access the sources by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```
sources:
  skip_tls_verify: true
```

Enabling TLS for Accelerator Server

To enable TLS for the Accelerator Server, the following properties must be provided in the `accelerator` section of the `tap-values.yaml` file:

```
server:
  tls:
    enabled: true
    key: SERVER-PRIVATE-KEY
    crt: SERVER-CERTIFICATE
```

Where:

- `SERVER-PRIVATE-KEY` is the pem encoded server private key.
- `SERVER-CERTIFICATE` is the pem encoded server certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Server:

```
server:
  tls:
    enabled: true
```

```
    key: |
      -----BEGIN PRIVATE KEY-----
      .
      .  < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      .  < certificate data >
      .
      -----END CERTIFICATE-----
```

Configuring skipping TLS verification of Engine calls for Accelerator Server

If you configure the Accelerator Engine to use TLS and use custom certificates, then you can configure the Accelerator Server to skip the TLS verification for calls to the Engine by providing the following property in the `accelerator` section of the `tap-values.yaml` file:

```
server:
  engine_skip_tls_verify: true
```

Enabling TLS for Accelerator Engine

To enable TLS for the Accelerator Engine, the following properties are provided in the `accelerator` section of the `tap-values.yaml` file:

```
engine:
  tls:
    enabled: true
    key: ENGINE-PRIVATE-KEY
    crt: ENGINE-CERTIFICATE
```

Where:

- `ENGINE-PRIVATE-KEY` is the pem encoded acc-engine private key.
- `ENGINE-CERTIFICATE` is the pem encoded acc-engine certificate.

Here is a sample `tap-values.yaml` configuration with TLS enabled for Accelerators Engine:

```
engine:
  tls:
    enabled: true
    key: |
      -----BEGIN PRIVATE KEY-----
      .
      .  < private key data >
      .
      -----END PRIVATE KEY-----
    crt: |
      -----BEGIN CERTIFICATE-----
      .
      .  < certificate data >
      .
      -----END CERTIFICATE-----
```

Next steps

- Creating accelerators

Create accelerators

This topic tells you how to create an accelerator in Tanzu Application Platform GUI.

An accelerator contains your conforming code and configurations that developers can use to create new projects that by default follow the standards defined in your accelerators.

Prerequisites

The following prerequisites are required to create an accelerator:

- Application Accelerator is installed. For information about installing Application Accelerator, see Installing Application Accelerator for VMware Tanzu.
- You can access Tanzu Application Platform GUI from a browser or use the Application Accelerator extension for VS Code.
    - For more information about Tanzu Application Platform GUI, see Overview of Tanzu Application Platform GUI.
    - For more information about Application Accelerator extension for VS Code, see Application Accelerator Visual Studio Code extension.
- kubectl is installed and authenticated with admin rights for your target cluster.

Getting started

You can use any Git repository to create an accelerator. You need the URL of the repository to create an accelerator.

For this example, the Git repository is `public` and contains a `README.md` file. These are options available when you create repositories on GitHub.

Use the following procedure to create an accelerator based on this Git repository:

1. Clone your Git repository.
2. Create a file named `accelerator.yaml` in the root directory of this Git repository.
3. Add the following content to the `accelerator.yaml` file:

    ```
    accelerator:
      displayName: Simple Accelerator
      description: Contains just a README
      iconUrl: https://images.freecreatives.com/wp-content/uploads/2015/05/smiley-5
    59124_640.jpg
      tags:
      - simple
      - getting-started
    ```

    Feel free to use a different icon if it uses a reachable URL.
4. Add the new `accelerator.yaml` file, commit this change and push to your Git repository.

Publishing the new accelerator

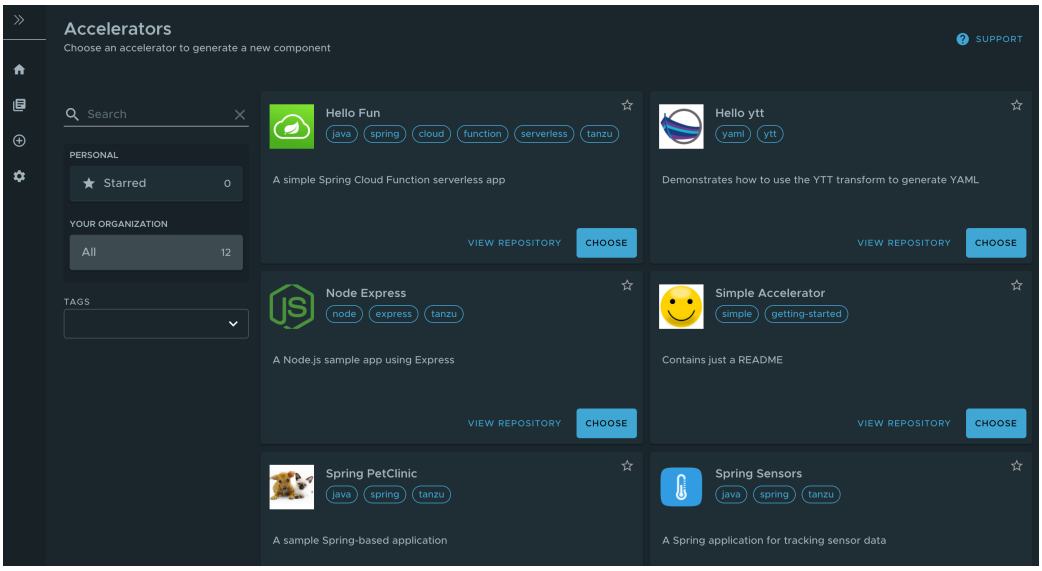
1. To publish your new accelerator, run:

    ```
    tanzu accelerator create simple --git-repository ${GIT_REPOSITORY_URL} --git-br
    anch ${GIT_REPOSITORY_BRANCH}
    ```

Where:

- `GIT-REPOSITORY-URL` is the URL for your Git repository where the accelerator is located.
- `GIT-REPOSITORY-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.

2. Refresh Tanzu Application Platform GUI or the Application Accelerator extension in VS Code to reveal the newly published accelerator. It might take a few seconds to refresh the catalog and add an entry for your new accelerator.



Alternatively, use the Tanzu CLI to create a separate manifest file and apply it to the cluster.

1. Create a `simple-manifest.yaml` file and add the following content, filling in with your Git repository and branch values.

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: simple
  namespace: accelerator-system
spec:
  git:
    url: YOUR-GIT-REPOSITORY-URL
    ref:
      branch: YOUR-GIT-BRANCH
```

2. To apply the `simple-manifest.yaml`, run this command in your terminal in the directory where you created this file:

```
tanzu accelerator apply -f simple-manifest.yaml
```

Using local-path for publishing accelerators

You can publish an accelerator directly from a local directory on your system. This helps when authoring accelerators and allows you to avoid having to commit every small change to a remote Git repository. You can also specify `--interval` so the accelerator is reconciled quicker when VMware push new changes.

```
tanzu accelerator create simple --local-path ${ACCELERATOR_PATH} --source-image ${SOUR
CE_IMAGE_REPO} --interval 10s
```

Where:

- `ACCELERATOR-PATH` is the path to the accelerator source. It is a fully qualified or a relative path. If your current directory is already the directory where your source is, then use ".".
- `SOURCE-IMAGE-REPO` is the name of the OCI image repository where you want to push the new accelerator source. If using Docker Hub, use something such as `docker.io/YOUR_DOCKER_ID/simple-accelerator-source`.

After you have made any additional changes, you can push the latest to the same OCI image repository using:

```
tanzu accelerator push --local-path ${ACCELERATOR_PATH} --source-image ${SOURCE_IMAGE_
REPO}
```

The accelerator now reflects the new content after approximately a 10-second wait as specified in the previous command.

Using accelerator fragments

Accelerator fragments are reusable accelerator components that can provide options, files, or transforms. They can be imported from accelerators using an `import` entry and the transforms from the fragment can be referenced in an `InvokeFragment` transform in the accelerator that is declaring the import. For additional details see InvokeFragment transform.

The accelerator samples include three fragments - `java-version`, `tap-initialize`, and `live-update`. See vmware-tanzu/application-accelerator-samples/fragments Git repository for the content of these fragments.

To discover what fragments are available to use, run:

```
tanzu accelerator fragment list
```

Look a the `java-version` fragment as an example. It contains the following `accelerator.yaml` file:

```
accelerator:
  options:
  - name: javaVersion
    inputType: select
    label: Java version to use
    choices:
    - value: "1.8"
      text: Java 8
    - value: "11"
      text: Java 11
    - value: "17"
      text: Java 17
    defaultValue: "11"
    required: true

engine:
  merge:
    - include: [ "pom.xml" ]
      chain:
      - type: ReplaceText
        regex:
          pattern: "<java.version>.*<"
          with: "'<java.version>' + #javaVersion + '<'"
    - include: [ "build.gradle" ]
      chain:
      - type: ReplaceText
        regex:
```

```
          pattern: "sourceCompatibility = .*"
          with: "'sourceCompatibility = ''' + #javaVersion + '''"
    - include: [ "config/workload.yaml" ]
      chain:
      - type: ReplaceText
        condition: "#javaVersion == '17'"
        substitutions:
          - text: "spec:"
            with: "'spec:\n  build:\n    env:\n    - name: BP_JVM_VERSION\n      valu
e: \"17\"'"
```

This fragment contributes the following to any accelerator that imports it:

1. An option named `javaVersion` with three choices `Java 8`, `Java 11`, and `Java 17`
2. Three `ReplaceText` transforms:
    - If the accelerator has a `pom.xml` file, then what is specified for `<java.version>` is replaced with the chosen version.
    - If the accelerator has a `build.gradle` file, then what is specified for `sourceCompatibility` is replaced with the chosen version.
    - If the accelerator has a `config/workload.yaml` file, and the user selected “Java 17” then a build environment entry of BP_JVM_VERSION is inserted into the `spec:` section.

Deploying accelerator fragments

To deploy new fragments to the accelerator system, use the new `tanzu accelerator fragment create` CLI command or apply a custom resource manifest file with either `kubectl apply` or the `tanzu accelerator apply` commands.

The resource manifest for the `java-version` fragment looks like this:

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Fragment
metadata:
  name: java-version
  namespace: accelerator-system
spec:
  displayName: Select Java Version
  git:
    ref:
      tag: GIT_TAG_VERSION
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example, `tap-1.4.0` is a valid Git tag for the `java-version` fragment.

To create the fragment, save the above manifest in a `java-version.yaml` file) and run:

```
tanzu accelerator apply -f ./java-version.yaml
```

Note

The `accelerator apply` command can be used to apply both Accelerator and Fragment resources.

To avoid having to create a separate manifest file, run:

```
tanzu accelerator fragment create java-version \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples.git \
  --git-tag ${GIT_TAG_VERSION} \
  --git-sub-path fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example,`tap-1.4.0` is a valid Git tag for the `java-version` fragment.

Now you can use this `java-version` fragment in an accelerator:

```
accelerator:
  displayName: Hello Fragment
  description: A sample app
  tags:
  - java
  - spring
  - cloud
  - tanzu

  imports:
  - name: java-version

engine:
  merge:
    - include: ["**/*"]
    - type: InvokeFragment
      reference: java-version
```

The earlier accelerator imports the `java-version` which, as seen earlier, provides an option to select the Java version to use for the project. It then instructs the engine to run the transforms provided in the fragment that updates the Java version used in `pom.xml` or `build.gradle` files from the accelerator.

For more detail on the use of fragments, see InvokeFragment transform.

Next steps

Learn how to:

- Write an accelerator.yaml.
- Configure accelerators with Accelerator Custom Resources.
- Manipulate files using Transforms.
- Use SpEL in the accelerator.yaml file.

Create accelerators

This topic tells you how to create an accelerator in Tanzu Application Platform GUI.

An accelerator contains your conforming code and configurations that developers can use to create new projects that by default follow the standards defined in your accelerators.

Prerequisites

The following prerequisites are required to create an accelerator:

- Application Accelerator is installed. For information about installing Application Accelerator, see Installing Application Accelerator for VMware Tanzu.

- You can access Tanzu Application Platform GUI from a browser or use the Application Accelerator extension for VS Code.
    - For more information about Tanzu Application Platform GUI, see Overview of Tanzu Application Platform GUI.
    - For more information about Application Accelerator extension for VS Code, see Application Accelerator Visual Studio Code extension.
- kubectl is installed and authenticated with admin rights for your target cluster.

Getting started

You can use any Git repository to create an accelerator. You need the URL of the repository to create an accelerator.

For this example, the Git repository is `public` and contains a `README.md` file. These are options available when you create repositories on GitHub.

Use the following procedure to create an accelerator based on this Git repository:

1. Clone your Git repository.
2. Create a file named `accelerator.yaml` in the root directory of this Git repository.
3. Add the following content to the `accelerator.yaml` file:

    ```
    accelerator:
      displayName: Simple Accelerator
      description: Contains just a README
      iconUrl: https://images.freecreatives.com/wp-content/uploads/2015/05/smiley-5
    59124_640.jpg
      tags:
      - simple
      - getting-started
    ```

    Feel free to use a different icon if it uses a reachable URL.
4. Add the new `accelerator.yaml` file, commit this change and push to your Git repository.

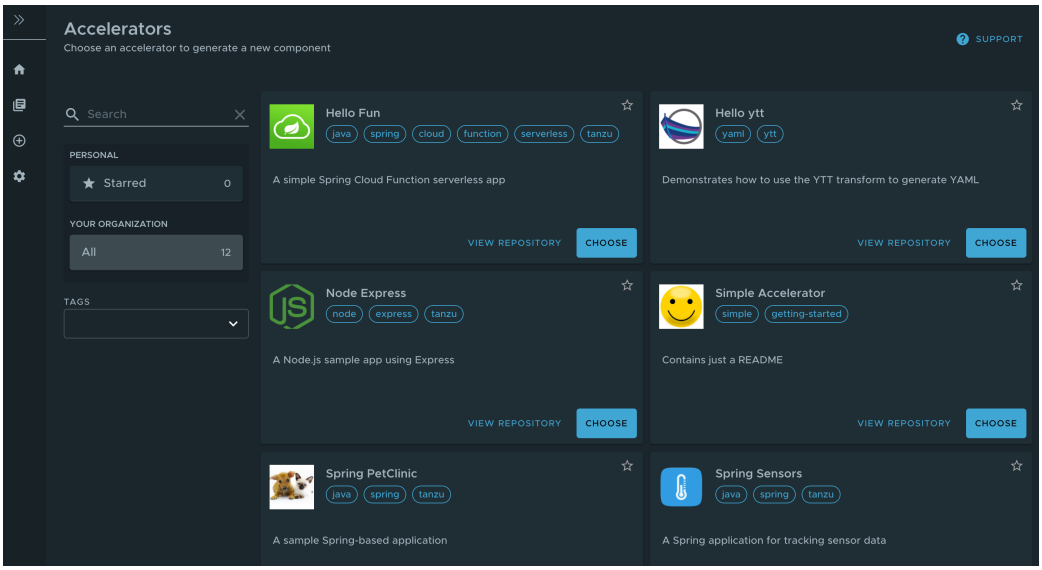
Publishing the new accelerator

1. To publish your new accelerator, run:

    ```
    tanzu accelerator create simple --git-repository ${GIT_REPOSITORY_URL} --git-br
    anch ${GIT_REPOSITORY_BRANCH}
    ```

    Where:

    - `GIT-REPOSITORY-URL` is the URL for your Git repository where the accelerator is located.
    - `GIT-REPOSITORY-BRANCH` is the name of the branch where you pushed the new `accelerator.yaml` file.
2. Refresh Tanzu Application Platform GUI or the Application Accelerator extension in VS Code to reveal the newly published accelerator. It might take a few seconds to refresh the catalog and add an entry for your new accelerator.



Alternatively, use the Tanzu CLI to create a separate manifest file and apply it to the cluster.

1. Create a `simple-manifest.yaml` file and add the following content, filling in with your Git repository and branch values.

   ```
   apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
   kind: Accelerator
   metadata:
     name: simple
     namespace: accelerator-system
   spec:
     git:
       url: YOUR-GIT-REPOSITORY-URL
       ref:
         branch: YOUR-GIT-BRANCH
   ```

2. To apply the `simple-manifest.yaml`, run this command in your terminal in the directory where you created this file:

   ```
   tanzu accelerator apply -f simple-manifest.yaml
   ```

Using local-path for publishing accelerators

You can publish an accelerator directly from a local directory on your system. This helps when authoring accelerators and allows you to avoid having to commit every small change to a remote Git repository. You can also specify `--interval` so the accelerator is reconciled quicker when VMware push new changes.

```
tanzu accelerator create simple --local-path ${ACCELERATOR_PATH} --source-image ${SOUR
CE_IMAGE_REPO} --interval 10s
```

Where:

- `ACCELERATOR-PATH` is the path to the accelerator source. It is a fully qualified or a relative path. If your current directory is already the directory where your source is, then use ".".
- `SOURCE-IMAGE-REPO` is the name of the OCI image repository where you want to push the new accelerator source. If using Docker Hub, use something such as `docker.io/YOUR_DOCKER_ID/simple-accelerator-source`.

After you have made any additional changes, you can push the latest to the same OCI image repository using:

```
tanzu accelerator push --local-path ${ACCELERATOR_PATH} --source-image ${SOURCE_IMAGE_
REPO}
```

The accelerator now reflects the new content after approximately a 10-second wait as specified in the previous command.

Using accelerator fragments

Accelerator fragments are reusable accelerator components that can provide options, files, or transforms. They can be imported from accelerators using an `import` entry and the transforms from the fragment can be referenced in an `InvokeFragment` transform in the accelerator that is declaring the import. For additional details see InvokeFragment transform.

The accelerator samples include three fragments - `java-version`, `tap-initialize`, and `live-update`. See vmware-tanzu/application-accelerator-samples/fragments Git repository for the content of these fragments.

To discover what fragments are available to use, run:

```
tanzu accelerator fragment list
```

Look a the `java-version` fragment as an example. It contains the following `accelerator.yaml` file:

```
accelerator:
  options:
  - name: javaVersion
    inputType: select
    label: Java version to use
    choices:
    - value: "1.8"
      text: Java 8
    - value: "11"
      text: Java 11
    - value: "17"
      text: Java 17
    defaultValue: "11"
    required: true

engine:
  merge:
    - include: [ "pom.xml" ]
      chain:
      - type: ReplaceText
        regex:
          pattern: "<java.version>.*<"
          with: "'<java.version>' + #javaVersion + '<'"
    - include: [ "build.gradle" ]
      chain:
      - type: ReplaceText
        regex:
          pattern: "sourceCompatibility = .*"
          with: "'sourceCompatibility = ''' + #javaVersion + '''\'"
    - include: [ "config/workload.yaml" ]
      chain:
      - type: ReplaceText
        condition: "#javaVersion == '17'"
        substitutions:
          - text: "spec:"
            with: "'spec:\n  build:\n    env:\n    - name: BP_JVM_VERSION\n      valu
e: \"17\"'"
```

This fragment contributes the following to any accelerator that imports it:

1. An option named `javaVersion` with three choices `Java 8`, `Java 11`, and `Java 17`
2. Three `ReplaceText` transforms:
    - If the accelerator has a `pom.xml` file, then what is specified for `<java.version>` is replaced with the chosen version.
    - If the accelerator has a `build.gradle` file, then what is specified for `sourceCompatibility` is replaced with the chosen version.
    - If the accelerator has a `config/workload.yaml` file, and the user selected “Java 17” then a build environment entry of BP_JVM_VERSION is inserted into the `spec:` section.

Deploying accelerator fragments

To deploy new fragments to the accelerator system, use the new `tanzu accelerator fragment create` CLI command or apply a custom resource manifest file with either `kubectl apply` or the `tanzu accelerator apply` commands.

The resource manifest for the `java-version` fragment looks like this:

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Fragment
metadata:
  name: java-version
  namespace: accelerator-system
spec:
  displayName: Select Java Version
  git:
    ref:
      tag: GIT_TAG_VERSION
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    subPath: fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example, `tap-1.4.0` is a valid Git tag for the `java-version` fragment.

To create the fragment, save the above manifest in a `java-version.yaml` file) and run:

```
tanzu accelerator apply -f ./java-version.yaml
```

Note

The `accelerator apply` command can be used to apply both Accelerator and Fragment resources.

To avoid having to create a separate manifest file, run:

```
tanzu accelerator fragment create java-version \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples.git \
  --git-tag ${GIT_TAG_VERSION} \
  --git-sub-path fragments/java-version
```

Where `GIT-TAG-VERSION` is the Git tag of the `java-version` fragment. For example,`tap-1.4.0` is a valid Git tag for the `java-version` fragment.

Now you can use this `java-version` fragment in an accelerator:

```
accelerator:
  displayName: Hello Fragment
  description: A sample app
  tags:
  - java
  - spring
  - cloud
  - tanzu

  imports:
  - name: java-version

engine:
  merge:
    - include: ["**/*"]
    - type: InvokeFragment
      reference: java-version
```

The earlier accelerator imports the `java-version` which, as seen earlier, provides an option to select the Java version to use for the project. It then instructs the engine to run the transforms provided in the fragment that updates the Java version used in `pom.xml` or `build.gradle` files from the accelerator.

For more detail on the use of fragments, see InvokeFragment transform.

Next steps

Learn how to:

- Write an accelerator.yaml.
- Configure accelerators with Accelerator Custom Resources.
- Manipulate files using Transforms.
- Use SpEL in the accelerator.yaml file.

Create an accelerator.yaml file in Application Accelerator

This topic tells you how to use Application Accelerator to create an `accelerator.yaml` file in Tanzu Application Platform (commonly known as TAP).

By including an `accelerator.yaml` file in your Accelerator repository, you can declare input options that users fill in using a form in the UI. Those option values control processing by the template engine before it returns the zipped output files. For more information, see the Sample accelerator.

When there is no `accelerator.yaml`, the repository still works as an accelerator but the files are passed unmodified to users.

`accelerator.yaml` has two top-level sections: `accelerator` and `engine`.

Accelerator

This section documents how an accelerator is presented to users in the web UI. For example:

```
accelerator:
  displayName: Hello Fun
  description: A simple Spring Cloud Function serverless app
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.p
ng
  tags:
  - java
```

```
- spring

options:
- name: deploymentType
  inputType: select
  choices:
  - value: none
    text: Skip Kubernetes deployment
  - value: k8s-simple
    text: Kubernetes deployment and service
  - value: knative
    text: Knative service
  defaultValue: k8s-simple
  required: true
```

Accelerator metadata

These properties are in accelerator listings such as the web UI:

- **displayName**: A human-readable name.
- **description**: A more detailed description.
- **iconUrl**: A URL pointing to an icon image.
- **tags**: A list of tags used to filter accelerators.

Accelerator options

The list of options is passed to the UI to create input text boxes for each option.

The following option properties are used by both the UI and the engine.

- **name**: Each option must have a unique, camelCase name. The option value entered by a user is made available as a SPeL variable name. For example, `#deploymentType`.

  You can specify your own default name by including:

  ```
  options:
  - name: projectName
  label: Name
  inputType: text
  defaultValue: myname
  required: true
  ```

- **dataType**: Data types that work with the UI are:
  - `string`
  - `boolean`
  - `number`
  - Custom types defined in the accelerator `types` section
  - Arrays of these such as `[string]`, `[number]`, and so on.

  Most input types return a string, which is the default. Use Boolean values with `checkbox`.

- **defaultValue**: This literal value pre-populates the option. Ensure that it's type matches the dataType. For example, use `["text 1", "text 2"]` for the dataType `[string]`. Options without a `defaultValue` can trigger a processing error if the user doesn't provide a value for that option.
- **validationRegex**: When present, a regex validates the string representation of the option value *when set*. It doesn't apply when the value is blank. As a consequence, don't use the

regex to enforce prerequisites. See **required** for that purpose.

This regex is used in several layers in Application Accelerator, built using several technologies, for example, JavaScript and Java. So refrain from using “exotic” regex features. Also, the regex applies to portions of the value by default. That is, `[a-z ]+` matches `Hello world` despite the capital `H`. To apply it to the whole value (or just start/end), anchor it using `^` and `$`.

Finally, backslashes in a YAML string using double quotes must be escaped, so to match a number, write `validationRegex: "\\d+"` or use another string style.

The following option properties are for UI purposes only.

- **label**: A human-readable version of the `name` identifying the option.
- **description**: A tooltip to accompany the input.
- **inputType**:
  - `text`: The default input type.
  - `textarea`: Single text value with larger input allowing line breaks.
  - `checkbox`: Ideal for Boolean values or multi-value selection from choices.
  - `select`: Single-value selection from choices using a drop-down menu.
  - `radio`: Alternative single-value selection from choices using buttons.
- **choices**: This is a list of predefined choices. Users can select from the list in the UI. Choices are supported by `checkbox`, `select`, and `radio`. Each choice must have a `text` property for the displayed text, and a `value` property for the value that the form returns for that choice. The list is presented in the UI in the same order as it is declared in `accelerator.yaml`.
- **required**: `true` forces users to enter a value in the UI.
- **dependsOn**: This is a way to control visibility by specifying the `name` and optional `value` of another input option. When the other option has a value exactly equal to `value`, or `true` if no `value` is specified, then the option with `dependsOn` is visible. Otherwise, it is hidden. Ensure that the value matches the dataType of the `dependsOn` option. For example, a multi-value option (`dataType = [string]`) such as a `checkbox` uses `[matched-value]` to trigger another option when `matched-value` (and only `matched-value`) is selected. See the following section for more information about `dependsOn`.

DependsOn and multi-value dataType

`dependsOn` tests for strict equality, even for multi-valued options. This means that a multi-valued option must not be used to trigger several other options unfolding, one for each value. Instead, use several single-valued options:

Instead of

```
options:
  - name: toppings
    dataType: [string]
    inputType: checkbox
    choices:
      - value: vegetables
        text: Vegetables
      - value: meat
        text: Meat
        ...
  - name: vegType
    dependsOn:
      name: toppings
```

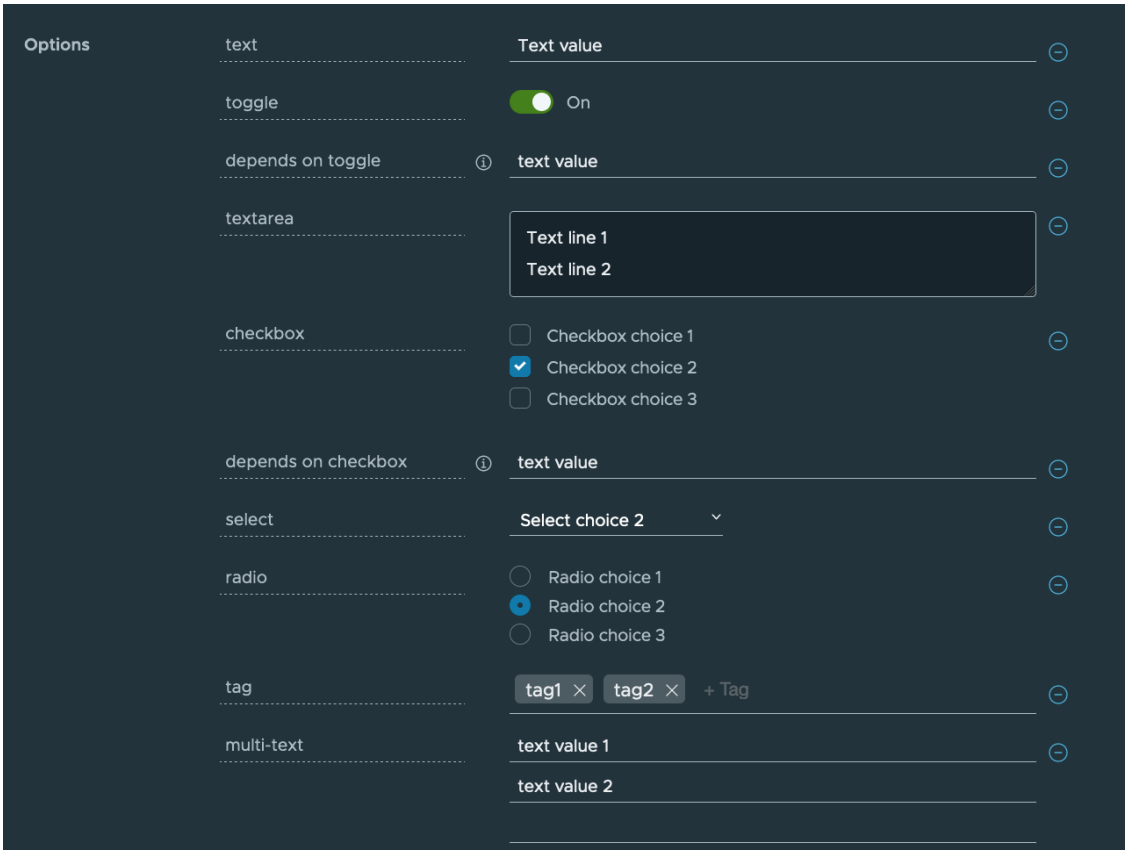
```
      value: [vegetables] # or vegetables, this won't do what you want either
  - name: meatType
    dependsOn:
      name: toppings
      value: [meat]
  ...
```

do this:

```
options:
  - name: useVeggies
    dataType: boolean
    inputType: checkbox
    label: Vegetables
  - name: useMeat
    dataType: boolean
    inputType: checkbox
    label: Meat
  - name: vegType
    dependsOn:
      name: useVeggies
      value: true
  - name: meatType
    dependsOn:
      name: useMeat
      value: true
  ...
```

Examples

The later screenshot and `accelerator.yaml` file snippet that follows demonstrates each `inputType`. You can also see the GitHub sample demo-input-types.



```
accelerator:
  displayName: Demo Input Types
  description: "Accelerator with options for each inputType"
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.p
ng
  tags: ["demo", "options"]

  options:

  - name: text
    display: true
    defaultValue: Text value

  - name: toggle
    display: true
    dataType: boolean
    defaultValue: true

  - name: dependsOnToggle
    label: 'depends on toggle'
    description: Visibility depends on the value of the toggle option being true.
    dependsOn:
      name: toggle
    defaultValue: text value

  - name: textarea
    inputType: textarea
    display: true
    defaultValue: |
      Text line 1
      Text line 2

  - name: checkbox
    inputType: checkbox
    display: true
    dataType: [string]
    defaultValue:
      - value-2
    choices:
      - text: Checkbox choice 1
        value: value-1
      - text: Checkbox choice 2
        value: value-2
      - text: Checkbox choice 3
        value: value-3

  - name: dependsOnCheckbox
    label: 'depends on checkbox'
    description: Visibility depends on the checkbox option containing exactly value va
lue-2.
    dependsOn:
      name: checkbox
      value: [value-2]
    defaultValue: text value

  - name: select
    inputType: select
    display: true
    defaultValue: value-2
    choices:
      - text: Select choice 1
        value: value-1
      - text: Select choice 2
        value: value-2
      - text: Select choice 3
        value: value-3
```

```
  - name: radio
    inputType: radio
    display: true
    defaultValue: value-2
    choices:
      - text: Radio choice 1
        value: value-1
      - text: Radio choice 2
        value: value-2
      - text: Radio choice 3
        value: value-3

engine:
  type: YTT
```

Engine

The engine section describes how to take the files from the accelerator root directory and transform them into the contents of a generated project file.

The YAML notation here defines what is called a transform. A transform is a function on a set of files. It uses a set of files as input. It produces a modified set of files as output derived from this input.

Different types of transforms do different tasks:

- Filtering the set of files: that is, removing, or keeping files that match certain criteria.
- Changing the contents of files. For example, replacing some strings in the files.
- Renaming or moving files: that is, changing the paths of the files.

The notation also provides the composition operators `merge` and `chain`, which enable you to create more complex transforms by composing simpler transforms together.

The following is an example of what is possible. To learn the notation, see Introduction to transforms.

Engine example

```
engine:
  include:
    ["**/*.md", "**/*.xml", "**/*.gradle", "**/*.java"]
  exclude:
    ["**/secret/**"]
  let:
    - name: includePoms
      expression:
        "#buildType == 'Maven'"
    - name: includeGradle
      expression: "#buildType == 'Gradle'"
  merge:
    - condition:
        "#includeGradle"
      include: ["*.gradle"]
    - condition: "#includePoms"
      include: ["pom.xml"]
    - include: ["**/*.java", "README.md"]
      chain:
        - type: ReplaceText
          substitutions:
            - text: "Hello World!"
              with: "#greeting"
```

```
  chain:
    - type: RewritePath
      regex: (.*)simpleboot(.*)
      rewriteTo: "#g1 + #packageName + #g2"
    - type: ReplaceText
      substitutions:
        - text: simpleboot
          with: "#packageName"
  onConflict:
    Fail
```

Engine notation descriptions

This section describes the notations in the preceding example.

`engine` is the global transformation node. It produces the final set of files to be zipped and returned from the accelerator. As input, it receives all the files from the accelerator repository root. The properties in this node dictate how this set of files is transformed into a final set of files zipped as the accelerator result.

`engine.include` filters the set of files, retaining only those matching a list of path patterns. This ensures that that the accelerator only detects files in the repository that match the list of patterns.

`engine.exclude` further restricts which files are detected. The example ensures files in any directory called `secret` are never detected.

`engine.let` defines additional variables and assigns them values. These derived symbols function such as options, but instead of being supplied from a UI widget, they are computed by the accelerator itself.

`engine.merge` executes each of its children in parallel. Each child receives a copy of the current set of input files. These are files remaining after applying the `include` and `exclude` filters. Each of the children therefore produces a set of files. All the files from all the children are then combined, as if overlaid on top of each other in the same directory. If more than one child produces a file with the same path, the transform resolves the conflict by dropping the file contents from the earlier child and keeping the contents from the later child.

`engine.merge.chain` specifies additional transformations to apply to the set of files produced by this child. In the example, `ReplaceText` is only applied to Java files and `README.md`.

`engine.chain` applies transformation to all files globally. The chain has a list of child transformations. These transformations are applied after everything else in the same node. This is the global node. The children in a chain are applied sequentially.

`engine.onConflict` specifies how conflict is handled when an operation, such as merging, produces multiple files at the same path: - `Fail` raises an error when there is a conflict. - `UseFirst` keeps the contents of the first file. - `UseLast` keeps the contents of the last file. - `Append` keeps both by using `cat <first-file> <second-file>`.

Advanced accelerator use

Additional advanced features can be leveraged when writing an `accelerator.yaml`. For more information see, Creating dynamic parameters using custom types

Application Accelerator sample accelerator.yaml file

This topic provides you with a sample accelerator file to get you started writing your own accelerators in Tanzu Application Platform (commonly known as TAP).

```
accelerator:
  # The `accelerator` section serves to document how an accelerator is presented to th
```

```
e
  # user in the accelerator web UI.

  # displayName: a descriptive human-readable name. Make this short so as to look nice
  #              in a list of many accelerators shown to a user.
  displayName: Hello Spring Boot

  # description: a more detailed description that a user can see if they took a closer
  #              look at a particular accelerator.
  description: Simple Hello World Rest Service based on Spring Boot

  # iconUrl: Optional, a nice colorful, icon for your accelerator to make it stand out
visually.
  iconUrl: https://raw.githubusercontent.com/simple-starters/icons/master/icon-cloud.p
ng

  # tags: A list of classification tags. The UI allows users to search for accelerator
s based on tags
  tags:
    - Java
    - Spring
    - Function

  # options are parameters that can affect how the accelerator behaves.
  # The purpose of the options section is
  #   - to list all applicable options
  #   - describe each option in enough detail so that the UI can create
  #     a suitable input widget for it.
  options: # a list of options
    # a first option
    - name:
        greeting
        # name: each option must have a name.
        # This must be
        #   - camelCase
        #   - unique (i.e. no two options can have the same name)
        # This is like a variable used by the accelerator to refer to
        # and use the value during its execution.
        # This name is internal to your accelerator and is not shown to
        # the user.
      label:
        Greeting Message
        # A human readable version of the `name`. This is used to identify an
        # option to the user in the UI.
        # This should be short (so as not to look ugly in a ui with limited
        # space available for labeling the input widgets).
        # There are no limits on what characters can be used in the label (so spaces
        # are allowed).
      description:
        Greeting message displayed by the Hello World app.
        # An optional more detailed description / explanation that can be shown to
        # to the user in the UI when the short label alone might not be enough to unde
rstand
        # its purpose.
      dataType:
        string
        # type of data the accelerator expects during execution (this is
        # like the type of the 'variable'.
        # possible dataTypes are string, boolean, number or [string] (the latter meani
ng a
        # list of strings
      inputType:
        text
        # Related to the dataType but somewhat independent, this identifies the type
        # of widget shown in the ui. Available types are:
        # - text - the default
```

```
        # - toggle (boolean on/off control)
        # - textarea (single text value with larger input that allows linebreaks)
        # - checkbox - multivalue selection from choices
        # - select - single value selection from choices
        # - radio - alternative single value selection from choices
        # - tag - multivalue input ui for entering single-word tags
      required: true
      defaultValue: Hello Accelerator
    # second option:
    - name: packageName
      label: "Package Name"
      description: Name of Java package
      dataType: string
      inputType: text
      defaultValue: somepackage
    # another option:
    - name: buildType
      label: Build Type
      description: Choose whether to use Maven or Gradle to build the project.
      dataType: string
      inputType: select
      choices:
        - value: Maven
          text: Maven (pom.xml)
        - value: Gradle
          text: Gradle (build.gradle)

# The 'engine' section describes how to take the files from the accelerator
# repo root folder and 'transform' them into the contents of a generated project / zi
p.
# transformation operate on the files as a set and can do things like:
# - filtering the set of files (i.e. removing / keeping only files that match certain
criteria)
# - changing the contents of a file (e.g. replacing some strings in them)
# - renaming or moving files (changing the paths of the files)
engine:
  # this is the 'global' transformation node. It produces the final set of
  # files to be zipped and returned from the accelerator.
  # As input it receives all the files from the accelerator repo root.

  # The properties in this node dictate how this set of files is
  # transformed into a final set of files to zip up as the accelerator
  # result.

  include:
    ["**/*.md", "**/*.xml", "**/*.gradle", "**/*.java"]
    # This globally defined `include` filters the set of files
    # retaining only those matching a given list of path patterns.
    # This can ensure that only files in the repo matching the list of
    # patterns will be seen / considered by the accelerator.

  exclude:
    ["**/secret/**"]
    # This globally defined `exclude` further restricts what files are considered.
    # This example ensures files in any directory called `secret` are never considere
d.

  # Under 'let' you can define additional variables and assign them values
  # These 'derived symbols' function much like options, but instead of
  # being supplied from a UI widget, they are computed by the accelerator itself.
  let:
    - name: includePoms # name of a symbol, must be camelCase
      expression:
        "#buildType == 'Maven'" # <- SpEL expression given as a string. You must take
care to use
        # proper quotes to avoid yaml treating '#' as starting a comment.
```

```
    - name: includeGradle
      expression: "#buildType == 'Gradle'"
  merge: # This merge section executes each of its children 'in parallel'.
    # Each child receives a copy of the current set of input files.
    # (i.e. the files that are remaining after considering the `include` and `exclude
`.
    # Each of the children thus produces a set of files.
    # Merge then combines all the files from all the children, as if by overlaying the
m on top of each other
    # in the same directory. If more than one child produces a file with the same pat
h,
    # this 'conflict' is resolved by dropping the file contents from the earlier child
    # and keeping only the later one.
    # merge child 1: this child node wants to contribute 'gradle' files to the final r
esult
    - condition:
        "#includeGradle" # this child is deactivated if the Gradle option was not sele
cted by the user
        # A deactivated child doesn't contribute anything to the final result.
      include: ["*.gradle"] # this child only focusses on gradle files (ignoring all o
ther files)
    # merge child 2: this child wants to contribute 'pom' files to the final result
    - condition: "#includePoms"
      include: ["pom.xml"]
    # merge child 3: this child wants to contribute Java code and README.md to the fin
al result
    - include: ["**/*.java", "README.md"]
      # Using: chain you can specify additional transformations to be applied to the s
et
      # of files produced by this child (i.e. the `ReplaceText` below is only applied
to .java files and README.md)
      chain:
        - type: ReplaceText
          substitutions:
            - text: "Hello World!"
              with: "#greeting"
  chain:
    # Globally specified chain, works like the one `from merge child 3`. But because i
t is global, it
    # applies transformation to all files globally.
    #
    # The chain has a list of child transformations. These transformation are applied
after everything else
    # in the same node (here we are in the 'global node').
    #
    # The children in a chain are applied sequentially.
    - type: RewritePath
      regex: (.*)simpleboot(.*)
      rewriteTo: "#g1 + #packageName + #g2" # SpEL expression. You can use '#g1' and
'#g2' to reference 'match groups'
    - type: ReplaceText
      substitutions:
        - text: simpleboot
          with: "#packageName"
  onConflict:
    Fail # other values are `UseFirst`, `UseLast`, or `Append`
    # when merging (or really any operation) produces multiple files at the same path
    # this defines how that conflict is handled.
    # Fail: raise an error when conflict happens
    # UseFirst: keep the contents of the first file
    # UseLast: keep the contents of the last file
    # Append: keep both as by using `cat <first-file> <second-file>`).
```

Use transforms in Application Accelerator

This topic tells you about using transforms with Application Accelerator.

When the accelerator engine executes the accelerator, it produces a ZIP file containing a set of files. The purpose of the `engine` section is to describe precisely how the contents of that ZIP file is created.

```
accelerator:
  ...
engine:
  <transform-definition>
```

Why transforms?

When you run an accelerator, the contents of the accelerator produce the result. It is made up of subsets of the files taken from the accelerator `<root>` directory and its subdirectories. You can copy the files as is, or transform them in a number of ways before adding them to the result.

The YAML notation in the `engine` section defines a transformation that takes as input a set of files (in the `<root>` directory of the accelerator) and produces as output another set of files, which are put into the ZIP file.

Every transform has a `type`. Different types of transform have different behaviors and different YAML properties that control precisely what they do.

In the following example, a transform of type `Include` is a filter. It takes as input a set of files and produces as output a subset of those files, retaining only those files whose path matches any one of a list of `patterns`.

If the accelerator has something like this:

```
engine:
  type: Include
  patterns: ['**/*.java']
```

This accelerator produces a ZIP file containing all the `.java` files from the accelerator `<root>` or its subdirectories but nothing else.

Transforms can also operate on the contents of a file, instead of merely selecting it for inclusion.

For example:

```
type: ReplaceText
substitutions:
- text: hello-fun
  with: "#artifactId"
```

This transform looks for all instances of a string `hello-fun` in all its input files and replaces them with an `artifactId`, which is the result of evaluating a SpEL expression.

Combining transforms

From the preceding examples, you can see that transforms such as `ReplaceText` and `Include` are too primitive to be useful by themselves. They are meant to be the building blocks of more complex accelerators.

To combine transforms, provide two operators called `Chain` and `Merge`. These operators are recursive in the sense that they compose a number of child transforms to create a more complex transform. This allows building arbitrarily deep and complex trees of nested transform definitions.

The following example shows what each of these two operators does and how they are used together.

Chain

Because transforms are functions whose input and output are of the same type (a set of files), you can take the output of one function and feed it as input to another. This is what `Chain` does. In mathematical terms, `Chain` is function composition.

You might, for example, want to do this with the `ReplaceText` transform. Used by itself, it replaces text strings in all the accelerator input files. What if you wanted to apply this replacement to only a subset of the files? You can use an `Include` filter to select only a subset of files of interest and chain that subset into `ReplaceText`.

For example:

```
type: Chain
transformations:
- type: Include
  patterns: ['**/pom.xml']
- type: ReplaceText
  substitutions:
  - text: hello-fun
    with: "#artifactId"
```

Merge

Chaining `Include` into `ReplaceText` limits the scope of `ReplaceText` to a subset of the input files. It also eliminates all other files from the result.

For example:

```
engine:
  type: Chain
  transformations:
  - type: Include
    patterns: ['**/pom.xml']
  - type: ReplaceText
    substitutions:
    - text: hello-fun
      with: "#artifactId"
```

The preceding accelerator produces a ZIP file that only contains `pom.xml` files and nothing else.

What if you also wanted other files in that ZIP? Perhaps you want to include some Java files as well, but don't want to apply the same text replacement to them.

You might be tempted to write something such as:

```
engine:
  type: Chain
  transformations:
  - type: Include
    patterns: ['**/pom.xml']
  - type: ReplaceText
    ...
  - type: Include
    patterns: ['**/*.java']
```

However, that doesn't work. If you chain non-overlapping includes together like this, the result is an empty result set. The reason is that the first include retains only `pom.xml` files. These files are fed

to the next transform in the chain. The second include only retains `.java` files, but because there are only `pom.xml` files left in the input, the result is an empty set.

This is where `Merge` comes in. A `Merge` takes the outputs of several transforms executed independently on the same input sourceset and combines or merges them together into a single sourceset.

For example:

```
engine:
  type: Merge
  sources:
  - type: Chain
    - type: Include
      patterns: ['**/pom.xml']
    - type: ReplaceText
      ...
  - type: Include
    patterns: ['**/*.java']
```

The preceding accelerator produces a result that includes both:

- The `pom.xml` files with some text replacements applied to them.
- Verbatim copies of all the `.java` files.

Shortened notation

It becomes cumbersome and verbose to combine transforms such as `Include`, `Exclude`, and `ReplaceText` with explicit `Chain` and `Merge` operators. Also, there is a common composition pattern to using them. Specifically, select an interesting subset using includes and excludes, apply a chain of additional transformations to the subset, and merge the result with the results of other transforms. That is why there is a transform known the `Combo` transform that combines `Include`, `Exclude`, `Merge`, and `Chain`.

For example:

```
type: Combo
include: ['**/*.txt', '**/*.md']
exclude: ['**/secret/*']
merge:
- <transform-definition>
- ...
chain:
- <transform-definition>
- ...
```

Each of the properties in this `Combo` transform is optional if you specify at least one.

Notice how each of the properties `include`, `exclude`, `merge`, and `chain` corresponds to the name of a type of transform, only spelled with lowercase letters.

If you specify only one of the properties, the `Combo` transform behaves exactly as if you used that type of transformation by itself.

For example:

```
merge: ...
```

Behaves the same as:

```
type: Merge
```

```
sources: ...
```

When you do specify multiple properties at the same time, the `Combo` transform composes them together and combines `Merge` and `Chain`.

For example:

```
include: ['**/*.txt', '**.md']
chain:
- type: ReplaceText
  ...
```

Is the same as:

```
type: Chain
transformations:
- type: Include
  patterns: ['**/*.txt', '**.md']
- type: Chain
  transformations:
  - type: ReplaceText
    ...
```

When you use all of the properties of `Combo` at once:

```
include: I
exclude: E
merge:
- S1
- S2
chain:
- T1
- T2
```

This is equivalent to:

```
type: Chain
transformations:
- type: Include
  patterns: I
- type: Exclude
  patterns: E
- type: Merge
  sources:
  - S1
  - S2
- T1
- T2
```

A Combo of one?

You can use the `Combo` as a convenient shorthand for a single type of annotation. However, though you can use it to combine multiple types, and though that is its main purpose, that doesn't mean you have to.

For example:

```
include: ["**/*.java"]
```

This is a `Combo` transform (remember, `type: Combo` is optional), but rather than combining multiple types of transforms, it only defines the `include` property. This makes it behaves exactly as an `Include` transform:

```
type: Include
patterns: ["**/*.java"]
```

It is usually more convenient to use a `Combo` transform to denote a single `Include`, `Exclude`, `Chain`, or `Merge` transform, because it is slightly shorter to write it as a `Combo` than writing it with an explicit `type:` property.

A common pattern with merge transforms

It is a common and useful pattern to use merges with overlapping contents to apply a transformation to a subset of files and then replace these changed files within a bigger context.

For example:

```
engine:
  merge:
  - include: ["**/*"]
  - include: ["**/pom.xml"]
    chain:
    - type: ReplaceText
        subsitutions: ...
```

The preceding accelerator copies all files from accelerator `<root>` while applying some text replacements only to `pom.xml` files. Other files are copied verbatim.

Here in more detail is how this works:

- **Transform A** is applied to the files from accelerator `<root>`. It selects all files, including `pom.xml` files.
- **Transform B** is *also* applied to the files from accelerator `<root>`. Again, `Merge` passes the same input independently to each of its child transforms. Transform B selects `pom.xml` files and replaces some text in them.

So both **Transform A** and **Transform B** output `pom.xml` files. The fact that both result sets contain the same file, and with different contents in them in this case, is a conflict that has to be resolved. By default, `Combo` follows a simple rule to resolve such conflicts: take the contents from the last child. Essentially, it behaves as if you overlaid both result sets one after another into the same location. The contents of the latter overwrite any previous files placed there by the earlier.

In the preceding example, this means that while both **Transform A** and **Transform B** produce contents for `pom.xml`, the contents from **Transform B** “wins.” So you get the version of the `pom.xml` that has text replacements applied to it rather than the verbatim copy from **Transform A**.

Conditional transforms

Every `<transform-definition>` can have a `condition` attribute.

```
- condition: "#k8sConfig == 'k8s-resource-simple'"
  include: [ "kubernetes/app/*.yaml" ]
  chain:
    - type: ReplaceText
      substitutions:
      - text: hello-fun
        with: "#artifactId"
```

When a transform’s condition is `false`, that transform is deactivated. This means it is replaced by a transform that does nothing. However, doing nothing can have different meanings depending on the context:

- When in the context of a `Merge`, a deactivated transform behaves like something that returns an empty set. A `Merge` adds things together using a kind of union; adding an empty set to union essentially does nothing.
- When in the context of a `'Chain` however, a deactivated transform behaves like the `identity` function instead (that is, `lambda (x) => x`). When you chain functions together, a value is passed through all functions in succession. So each function in the chain has the chance to do something by returning a different modified value. If you are a function in a chain, to do nothing means to return the input you received unchanged as your output.

If a transform is deactivated in the context of your accelerator definition, it evaluates to false and is ignored. Your accelerator behaves as if you deleted or commented out that transform's YAML text from the accelerator definition file.

The following examples illustrate both cases.

Conditional 'Merge' transform

This example, **transform A**, has a conditional transform in a `Merge` context:

```
merge:
  - condition: "#k8sConfig == 'k8s-resource-simple'"
    include: [ "kubernetes/app/*.yaml" ]
    chain:
      ...
  - include: [ "pom.xml" ]
    chain:
      ...
```

If the condition of **transform A** is `false`, it is replaced with an empty set because it is used in a `Merge` context. This has the same effect as if the whole of **transform A** was deleted or commented out:

```
merge:
  - include: [ "pom.xml" ]
    chain:
      ...
```

In this example, if the condition is `false`, only `pom.xml` file is in the result.

Conditional 'Chain' transform

In the following example, some conditional transforms are used in a `Chain` context:

```
merge:
- include: [ '**/*.json' ]
  chain:
  - type: ReplaceText
    condition: '#customizeJson'
    substitutions: ...
  - type: JsonPrettyPrint
    condition: '#prettyJson'
    indent: '#jsonIndent'
```

In the preceding example, both **transform A** and **transform B** are conditional and used in a `Chain` context. **Transform A** is chained after the `include` transform. Whereas **transform B** is chained after **transform A**. When either of these conditions is `false`, the corresponding transform behaves like the identity function. Namely, whatever set of files it receives as input is exactly what it returns as output.

For example, if **transform A**'s condition is `false`, it behaves as if **transform A** isn't there. **Transform A** is chained after `include` so it receives the `include`'s result, returns it unchanged, and this is passed to **transform B**. In other words, the result of the `include` is passed as is to **transform B**.

A small gotcha with using conditionals in merge transforms

As mentioned earlier, it is a useful pattern to use merges with overlapping contents. But you must be careful using this in combination with conditional transforms.

For example:

```
engine:
  merge:
  - include: ["**/*"]
  - include: ["**/pom.xml"]
    chain:
    - type: ReplaceText
      subsitutions: ...
```

If you only want to include pom files, if you select a `useMaven` option, when you add a 'condition' to **transform B** to deactivate it, the final result still contains `pom.xml` files.:

```
engine:
  merge:
  - include: "**/*"
  - condition: '#useMaven'
    include: ["**/pom.xml"]
    chain:
    - type: ReplaceText
      subsitutions: ...
```

This is because if a transform is deactivated in the context of your accelerator definition, it evaluates to false and is ignored. So when `#useMaven` is `false`, the example reduces to:

```
engine:
  merge:
  - include: ["**/*"]
```

This accelerator copies all files from accelerator `<root>`, including `pom.xml`.

There are several ways to avoid this. One is to ensure the `pom.xml` files are not included in **transform A** by explicitly excluding them:

```
  ...
  - include: ["**/*"]
    exclude: ["**/pom.xml"]
  ...
```

Another way is to apply the exclusion of `pom.xml` conditionally in a `Chain` after the main transform:

```
engine:
  merge:
  - include: ["**/*"]
  - include: ["**/pom.xml"]
    chain:
    - type: ReplaceText
        subsitutions: ...
  chain:
  - condition: '!#useMaven'
    exclude: ['**/pom.xml']
```

Merge conflict

The representation of the set of files upon which transforms operate is richer than what you can physically store on a file system. A key difference is that in this case, the set of files allows for multiple files with the same path to exist at the same time. When files are initially read from a physical file system, or a ZIP file, this situation does not arise. However, as transforms are applied to this input, it can produce results that have more than one file with the same path and yet different contents.

Earlier examples illustrated this happening through a `merge` operation. For example:

```
merge:
- include: ["**/*"]
- include: ["**/pom.xml"]
  chain:
  - type: ReplaceText
    subsitutions: ...
```

The result of the preceding `merge` is two files with path `pom.xml`, assuming there was a `pom.xml` file in the input. **Transform A** produces a `pom.xml` that is a verbatim copy of the input file. **Transform B** produces a modified copy with some text replaced in it.

It is impossible to have two files on a disk with the same path. Therefore, this conflict must be resolved before you can write the result to disk or pack it into a ZIP file.

As the example shows, merges are likely to give rise to these conflicts, so you might call this a “merge conflict.” However, such conflicts can also arise from other operations. For example, `RewritePath`:

```
type: RewritePath
regex: '.*.md'
rewriteTo: "'docs/README.md'"
```

This example renames any `.md` file to `docs/README.md`. Assuming the input contains more than one `.md` file, the output contains multiple files with path `docs/README.md`. Again, this is a conflict, because there can only be one such file in a physical file system or ZIP file.

Resolving merge conflicts

By default, when a conflict arises, the engine doesn’t do anything with it. Our internal representation for a set of files allows for multiple files with the same path. The engine carries on manipulating the files as is. This isn’t a problem until the files must be written to disk or a ZIP file. If a conflict is still present at that time, an error is raised.

If your accelerator produces such conflicts, they must be resolved before writing files to disk. VMware provides the UniquePath transform. This transform allows you to specify what to do when more than one file has the same path. For example:

```
chain:
- type: RewritePath
  regex: '.*.md'
  rewriteTo: "'docs/README.md'"
- type: UniquePath
  strategy: Append
```

The result of the above transform is that all `.md` files are gathered up and concatenated into a single file at path `docs/README.md`. Another possible resolution strategy is to keep only the contents of one of the files. See Conflict Resolution.

Combo transform includes some convenient built-in support for conflict resolution. It automatically selects the `UseLast` strategy if none is explicitly supplied. You rarely, if ever, need to specify a conflict resolution strategy.

File ordering

As mentioned earlier, our set of files representation is richer than the files on a typical file system in that it allows for multiple files with the same path. Another way in which it is richer is that the files in the set are ordered. That is, a `FileSet` is more like an ordered list than an unordered set.

In most situations, the order of files in a `FileSet` doesn't matter. However, in conflict resolution it is significant. If you look at the preceding `RewritePath` example again, you might wonder about the order in which the various `.md` files are appended to each other. This ordering is determined by the order of the files in the input set.

So what is that order? In general, when files are read from disk to create a `FileSet`, you cannot assume a specific order. Yes, the files are read and processed in a sequential order, but the actual order is not well defined. It depends on implementation details of the underlying file system. The accelerator engine therefore does not ensure a specific order in this case. It only ensures that it preserves whatever ordering it receives from the file system, and processes files in accord with that order.

If you do not want the file order produced from reading directly from a file system and want to control the order of the sections in the `README.md` file, change the order of the merge children. `Merge` processes its children in order and reflects this order in the resulting output

For example:

```
chain:
  - merge:
      - include: ['README.md']
      - include: ['DEPLOYMENT.md']
        chain:
          - type: RewritePath
            rewriteTo: "'README.md'"
  - type: UniquePath
    strategy: Append
```

In this example, `README.md` from the first child of `merge` comes before `DEPLOYMENT.md` from the second child of `merge`.

Next steps

This introduction focused on an intuitive understanding of the `<transform-definition>` notation. This notation defines precisely how the accelerator engine generates new project content from the files in the accelerator root.

For more information, see:

- An exhaustive Reference of all built-in transform types
- A sample, commented accelerator.yaml to learn from a concrete example

Use custom types in Application Accelerator

This topic tells you how to declare new `types` in `accelerator.yaml`

Use these types for options declaration, in addition to the built-in types `string`, `number`, and `boolean`.

In `accelerator.yaml`, use the `types` entry (inside the top-level `accelerator` section) to define custom types.

The name must be an initial capital letter.

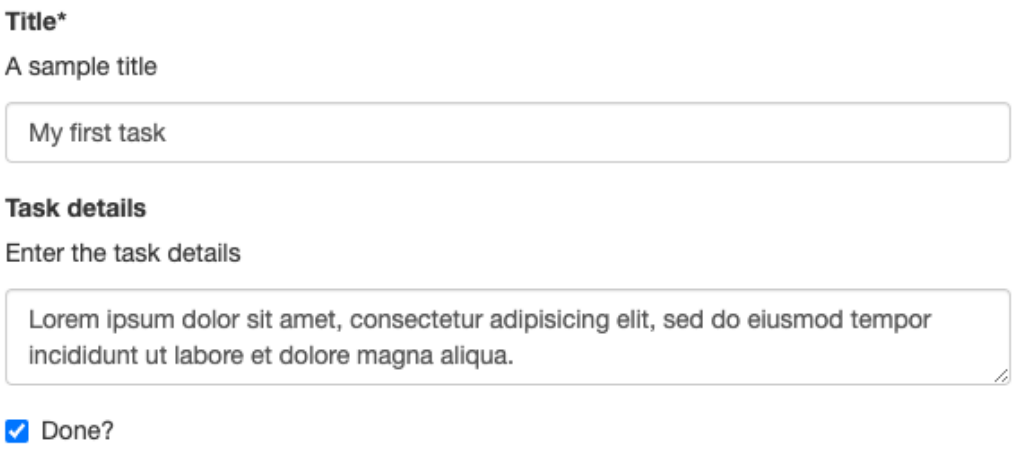
In the following example, the `struct` type definition is syntactically equivalent to a sequence of option definitions:

```
accelerator:
  options:
    ...
  types:
    - name: Task
      struct:
        - name: title
          dataType: string
          label: Title
          description: A sample title
        - name: details
          label: Task details
          description: Enter the task details
        - name: done
          dataType: boolean
          label: Done?
          defaultValue: false
```

This example creates a new *type* that is available for the `dataType` property of any option. For example,

```
accelerator:
  options:
    - name: myTask
      dataType: Task
  types:
    ...
```

UIs render similar to the following:



and associate the entered values to the `myTask` top-level name, resulting in the following example values submission (here represented using JSON notation):

```
{
  "myTask": { // Note the use of a nested object here
    "title": "Get job done!",
    "details": "Needs this asap",
    "done": false
```

```
  }
}
```

The type of the `myTask` value is `object` (in Javascript/JSON parlance) and `Map<String, ?>` when seen from the Java engine side.

The earlier example is technically possible with the custom types feature, but brings little benefit over having three options named to achieve the same result, for example, `myTaskTitle`, `myTaskDetails`, and `myTaskDone`. The value of custom types is when they are used in sequence types, allowing you to enter an unbounded list of structured data:

```
accelerator:
  options:
    - name: myTasks
      dataType: [Task]
  types:
    ...
```

Which might result in the following example submission (JSON):

```
{
  "myTasks": [ // Note the use of JSON array
    {  // with elem 0 being an object
      "details": "something",
      "done": true,
      "title": "The Title"
    },
    {  // and elem 1 as well, etc
      "details": "something else",
      "done": false,
      "title": "The other Title"
    }
  ]
}
```

Limitations

A `struct` custom type declaration is made of an ordered series of option definitions. The support and semantics for individual text boxes of option-definition-like elements when used in the type *declaration* are stated in the following example.

When *referencing* a custom type in an option definition, some previously valid properties of an option definition might become irrelevant or unsupported. This is stated in the following example:

```
accelerator:
  types:
    - name: MyType
      struct:
        - name: someField   # the "option name" will become a 'property' of the newly
created type
          dataType: string  # is the type of this single property. Typically, will be
a simple
                            # scalar type like string or number
          defaultValue: foo # supported and is the default if not overridden at usage
point by the option's defaultValue
          description: something # will become the description for the field's widget
          choices:              # supported
            - value: v
              text:  t
          validationRegex:      # validates that single property
          label:                # will become the "title" of the widget
          inputType:            # supported
```

```
          required:              # supported
          dependsOn:             # supported against other properties of THIS struct
    .. other fields
  options:
    - name: anOptionThatUsesACustomType
      dataType: MyType
      defaultValue: # supported, should then be an object (or array thereof)
      description:  # supported, is the description of the whole option (as opposed to
individual fields)
      label:        # supported, idem
      choices:      # NOT supported
        - value: v
          text:  t
      validationRegex: # NOT supported
      inputType:       # NOT supported
      required:        # technically supported, useless in practice
      dependsOn:       # OK to depend on another option
```

Interaction with SpEL

Everywhere that SpEL is used in the engine syntax, accelerator authors might use SpEL syntax for accessing properties or array elements:

```
#myTasks[2]['done']
```

Do not use array indexing either with a literal number or a variable, as the purpose of the list of the custom types feature is that you don't know the data length in advance. For more information about idiomatic uses of repeated structured data, see Loop Transform.

Interaction with Composition

Using composition alongside custom types has the following advantages/disadvantages:

- You might want to **leverage** types declared in an imported fragment
- There might be a type **name clash** between a host accelerator/fragment and an imported fragment, because the imported fragment author is unaware of how the fragment is to be used.

For more information about the syntax to customize the imported types names, see Use fragments in Application Accelerator.

Use fragments in Application Accelerator

This topic tells you how to use fragments in Application Accelerator.

Introduction

Despite their benefits, writing and maintaining, accelerators can become repetitive and verbose as new accelerators are added. Some create a project different from the next with similar aspects, requiring some form of copy-paste.

To alleviate this concern, Application Accelerators support a feature named Composition that allows the re-use of parts of an accelerator, called **fragments**.

Introducing fragments

A **fragment** looks exactly the same as an accelerator:

- It is made of a set of files.
- It contains an `accelerator.yaml` descriptor with options, declarations, and a root transform.

There are differences however. Namely:

- Fragments are declared to the system differently. They are filed as **fragment** custom resources.
- They deal with files differently. Because fragments deal with their own files and files from the accelerator using them, they use dedicated conflict resolution strategies (more on this later).

Fragments may be thought of as “functions” in programming languages. After being defined and referenced, they are “called” at various points in the main accelerator. The composition feature is designed with ease of use and “common use first” in mind, so these “functions” are typically called with as little noise as possible. You can also call them complex or different values.

Composition relies on two building blocks that play hand in hand:

- The `imports` section at the top of an accelerator manifest.
- The, `InvokeFragment` transform, to be used alongside any other transform.

| The `imports` section explained

To be usable in composition, a fragment MUST be *imported* into the dedicated section of an accelerator manifest:

```
accelerator:
  name: my-awesome-accelerator
  options:
    - name: flavor
      dataType: string
      defaultValue: Strawberry
  imports:
    - name: my-first-fragment
    - name: another-fragment
engine:
  ...
```

The effect of importing a fragment this way is twofold:

- It makes its files available to the engine (therefore importing a fragment is required).
- It exposes all of its options as options of the accelerator as if they were defined by the accelerator itself.

So in the earlier example, if the `my-first-fragment` fragment had the following `accelerator.yaml` file:

```
accelerator
  name: my-first-fragment
  options:
    - name: optionFromFragment
      dataType: boolean
      description: this option comes from the fragment
...
```

Then it is as if the `my-awesome-accelerator` accelerator defined it:

```
accelerator:
  name: my-awesome-accelerator
  options:
```

```
    - name: flavor
      dataType: string
      defaultValue: Strawberry
    - name: optionFromFragment
      dataType: boolean
      description: this option comes from the fragment
  imports:
    - name: my-first-fragment
    - name: another-fragment
engine:
  ...
```

All the metadata about options (type, default value, description, choices if applicable, *etc.*) come along when imported.

Because of this, users are prompted for a value for those options that come from fragments, as if they were options of the accelerator.

Using the `InvokeFragment` Transform

The second part at play in the composition is the `InvokeFragment` Transform.

As with any other transform, it may be used anywhere in the `engine` tree and receives files that are "visible" at that point. Those files, alongside those that make up the fragment, are made available to the fragment logic. If the fragment wants to choose between two versions of a file for a path, two new conflict resolution strategies are available: `FavorForeign` and `FavorOwn`.

The behavior of the `InvokeFragment` transform is very straightforward: after having validated options that the fragment expects (and maybe after having set default values for options that support one), it executes the whole transform of the fragment *as if it was written in place of* `InvokeFragment`.

See the `InvokeFragment` reference page for more explanations, examples, and configuration options. This topic now focuses on additional features of the `imports` section that are seldom used but still available to cover more complex use cases.

Back to the `imports` section

The complete definition of the `imports` section is as follows, with features in increasing order of "complexity":

```
accelerator:
  name: ...
  options:
    - name: ...
    ...
  imports:
    - name: some-fragment

    - name: another-fragment
      expose:
        - name: "*"
      exposeTypes:
        - name: "*"

    - name: yet-another-fragment
      expose:
        - name: someOption

        - name: someOtherOption
          as: aDifferentName
      exposeType:
        - name: SomeType
```

```
        - name: SomeOtherType
          as: ADifferentName
engine:
  ...
```

As shown earlier, the `imports` section calls a list of fragments to import. By default, all their options and types become options/type of the accelerator. Those options appear *after* the options defined by the accelerator, in the order the fragments are imported in.

It is even possible for a fragment to import another fragment, the semantics being the same as when an accelerator imports a fragment. This is a way to break apart a fragment even further if needed.

When importing a fragment, you can select which options of the fragment to make available as options of the accelerator. **This feature should only be used when a name clash arises in option names.**

The semantics of the `expose` block are as follows:

- For every `name`/`as` pair, don't use the original (`name`) of the option but instead, use the alias (`as`). Other metadata about the option is left unchanged.
- If the special `name: "*"` (which is NOT a legit option name usually) appears, all imported option names that are not remapped (the index at which the `*` appears in the YAML list is irrelevant) might be exposed with their original name.
- The default value for `expose` is `[{name: "*"}]`, that is, by default exposes all options with their original name.
- As soon as a single remap rule appears, the default is overridden. For example, to override some names AND expose the others unchanged, the `*` must be explicitly re-added.
- To explicitly un-expose ALL options from an imported fragment, an empty array may be used and overrides the default: `expose: []`.

Similarly, you can also select which custom types of the fragment to make available as types of the accelerator. **This feature should only be used when a name clash arises in types names.**

The semantics of the `exposeTypes` block are as follows:

- For every `name`/`as` pair, don't use the original (`name`) of the type but instead, use the alias (`as`). Options that used the original name are automatically "rewritten" to use the new name.
- If the special `name: "*"` appears, which is NOT usually a legit type name, all imported other type names that are not remapped are exposed with their original name. The index at which the `*` appears in the YAML list is irrelevant.
- The default value for `exposeTypes` is `[{name: "*"}]`, that is, by default exposes all types with their original name.
- As soon as a single remap rule appears, the default is overridden. For example, to override some names AND expose the others unchanged, the `*` must be explicitly re-added.
- To explicitly un-expose ALL types from an imported fragment, an empty array may be used, which overrides the default: `exposeTypes: []`.

Using `dependsOn` in the `imports` section

Lastly, as a convenience for the conditional use of fragments, you can make an exposed imported option *depend on* another option, as in the following example:

```
  imports:
    - name: tap-initialize
      expose:
        - name: gitRepository
          as: gitRepository
          dependsOn:
            name: deploymentType
            value: workload
        - name: gitBranch
          as: gitBranch
          dependsOn:
            name: deploymentType
            value: workload
```

This plays well with the use of `condition`, as in the following example:

```
...
engine:
  ...
    type: InvokeFragment
    condition: "#deploymentType == 'workload'"
    reference: tap-initialize```
```

Discovering fragments using Tanzu CLI accelerator plug-in

Using the accelerator plug-in for Tanzu CLI, you can view a list of available fragments. Run:

```
tanzu accelerator fragment list
```

To see a list of available accelerator fragments. For example:

```
NAME                                READY   REPOSITORY
app-sso-client                      true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/app-sso-client@sha256:ed5cf5544477d52d4c7baf3a76f71a11298
7856e77558697112e46947ada9241
java-version                        true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/java-version@sha256:df99a5ace9513dc8d083fb5547e2a24770dfb
08ec111b6591e98497a329b969d
live-update                         true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/live-update@sha256:c2eda015b0f811b0eeaa587b6f2c5410ac87d4
0701906a357cca0decb3f226a4
spring-boot-app-sso-auth-code-flow  true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/spring-boot-app-sso-auth-code-flow@sha256:78604c96dd52697
ea0397d3933b42f5f5c3659cbcdc0616ff2f57be558650499
tap-initialize                      true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/tap-initialize@sha256:7a3ae8f9277ef633200622dbf9d0f5a07de
a25351ac3dbf803ea2fa759e3baac
tap-workload                        true    source-image: dev.registry.tanzu.vmware.c
om/app-accelerator/fragments/tap-workload@sha256:8056ad9f05388883327d9bbe457e6a0122dc4
52709d179f683eceb6d848338d0
```

The `tanzu accelerator fragment get <fragment-name>` command shows all the options defined for the fragment and also any accelerators or other fragments that import this fragment. Run:

```
tanzu accelerator fragment get java-version
```

The following output is displayed:

```
name: java-version
namespace: accelerator-system
displayName: Select Java Version
```

```
ready: true
options:
- choices:
  - text: Java 8
    value: "1.8"
  - text: Java 11
    value: "11"
  - text: Java 17
    value: "17"
  defaultValue: "11"
  inputType: select
  label: Java version to use
  name: javaVersion
  required: true
artifact:
  message: Resolved revision: dev.registry.tanzu.vmware.com/app-accelerator/fragments/
java-version@sha256:df99a5ace9513dc8d083fb5547e2a24770dfb08ec111b6591e98497a329b969d
  ready: true
  url: http://source-controller-manager-artifact-service.source-system.svc.cluster.loc
al./imagerepository/accelerator-system/java-version-frag-97nwp/df99a5ace9513dc8d083fb5
547e2a24770dfb08ec111b6591e98497a329b969d.tar.gz
imports:
  None
importedBy:
  accelerator/java-rest-service
  accelerator/java-server-side-ui
  accelerator/spring-cloud-serverless
```

This shows the `options` and `importedBy` with a list of three accelerators that import this fragment.

Correspondingly, the `tanzu accelerator get <accelerator-name>` shows the fragments that an accelerator imports. Run:

```
tanzu accelerator get java-rest-service
```

The following output is shown:

```
name: java-rest-service
namespace: accelerator-system
description: A Spring Boot Restful web application including OpenAPI v3 document gener
ation and database persistence, based on a three-layer architecture.
displayName: Tanzu Java Restful Web App
iconUrl: ...abbreviated...
source:
  image: dev.registry.tanzu.vmware.com/app-accelerator/samples/java-rest-service@sha25
6:c098bb38b50d8bbead0a1b1e9be5118c4fdce3e260758533c38487b39ae0922d
  secret-ref: [{reg-creds}]
tags:
- java
- spring
- web
- jpa
- postgresql
- tanzu
ready: true
options:
- defaultValue: customer-profile
  inputType: text
  label: Module artifact name
  name: artifactId
  required: true
- defaultValue: com.example
  inputType: text
  label: Module group name
  name: groupId
```

```
  required: true
- defaultValue: com.example.customerprofile
  inputType: text
  label: Module root package
  name: packageName
  required: true
- defaultValue: customer-profile-database
  inputType: text
  label: Database Instance Name this Application will use (can be existing one in
    the cluster)
  name: databaseName
  required: true
- choices:
  - text: Maven (https://maven.apache.org/)
    value: maven
  - text: Gradle (https://gradle.org/)
    value: gradle
  defaultValue: maven
  inputType: select
  name: buildTool
  required: true
- choices:
  - text: Flyway (https://flywaydb.org/)
    value: flyway
  - text: Liquibase (https://docs.liquibase.com/)
    value: liquibase
  defaultValue: flyway
  inputType: select
  name: databaseMigrationTool
  required: true
- dataType: boolean
  defaultValue: false
  label: Expose OpenAPI endpoint?
  name: exposeOpenAPIEndpoint
- defaultValue: ""
  dependsOn:
    name: exposeOpenAPIEndpoint
  inputType: text
  label: System API Belongs To
  name: apiSystem
- defaultValue: ""
  dependsOn:
    name: exposeOpenAPIEndpoint
  inputType: text
  label: Owner of API
  name: apiOwner
- defaultValue: ""
  dependsOn:
    name: exposeOpenAPIEndpoint
  inputType: text
  label: API Description
  name: apiDescription
- choices:
  - text: Java 8
    value: "1.8"
  - text: Java 11
    value: "11"
  - text: Java 17
    value: "17"
  defaultValue: "11"
  inputType: select
  label: Java version to use
  name: javaVersion
  required: true
- dataType: boolean
  defaultValue: true
```

```
  dependsOn:
    name: buildTool
    value: maven
  inputType: checkbox
  label: Include TAP IDE Support for Java Workloads
  name: liveUpdateIDESupport
- defaultValue: dev.local
  dependsOn:
    name: liveUpdateIDESupport
  description: The prefix for the source image repository where source can be stored
    during development
  inputType: text
  label: The source image repository prefix to use when pushing the source
  name: sourceRepositoryPrefix
artifact:
  message: Resolved revision: dev.registry.tanzu.vmware.com/app-accelerator/samples/ja
va-rest-service@sha256:c098bb38b50d8bbead0a1b1e9be5118c4fdce3e260758533c38487b39ae0922
d
  ready: true
  url: http://source-controller-manager-artifact-service.source-system.svc.cluster.loc
al./imagerepository/accelerator-system/java-rest-service-acc-wcn8x/c098bb38b50d8bbead0
a1b1e9be5118c4fdce3e260758533c38487b39ae0922d.tar.gz
imports:
  java-version
  live-update
  tap-workload
```

The `imports` section at the end shows the fragments that this accelerator imports. The `options` section shows all options defined for this accelerator. This includes all options defined in the imported fragments, for example, the options for the Java version imported from the `java-version` fragment.

Transforms reference

This topic provides you with a list and brief description of the available Application Accelerator transforms in Tanzu Application Platform (commonly known as TAP).

Available transforms

You can use:

- Combo as a shortcut notation for many common operations. It combines the behaviors of many of the other transforms.
- Include to select files to operate on.
- Exclude to select files to operate on.
- Merge to work on subsets of inputs and to gather the results at the end.
- Chain to apply several transforms in sequence using function composition.
- Let to introduce new scoped variables to the model.
- InvokeFragment allows re-using various fragments across accelerators.
- ReplaceText to perform simple token replacement in text files.
- RewritePath to move files around using regular expression (regex) rules.
- OpenRewriteRecipe to apply Rewrite recipes, such as package rename.
- YTT to run the `ytt` tool on its input files and gather the result.
- UseEncoding to set the encoding to use when handling files as text.

- UniquePath to decide what to do when several files end up on the same path.
- Loop to iterate over a list and apply a transform for each element.
- Provenance to generate a manifest of the accelerator run.

See also

- Conflict Resolution

Transforms reference

This topic provides you with a list and brief description of the available Application Accelerator transforms in Tanzu Application Platform (commonly known as TAP).

Available transforms

You can use:

- Combo as a shortcut notation for many common operations. It combines the behaviors of many of the other transforms.
- Include to select files to operate on.
- Exclude to select files to operate on.
- Merge to work on subsets of inputs and to gather the results at the end.
- Chain to apply several transforms in sequence using function composition.
- Let to introduce new scoped variables to the model.
- InvokeFragment allows re-using various fragments across accelerators.
- ReplaceText to perform simple token replacement in text files.
- RewritePath to move files around using regular expression (regex) rules.
- OpenRewriteRecipe to apply Rewrite recipes, such as package rename.
- YTT to run the `ytt` tool on its input files and gather the result.
- UseEncoding to set the encoding to use when handling files as text.
- UniquePath to decide what to do when several files end up on the same path.
- Loop to iterate over a list and apply a transform for each element.
- Provenance to generate a manifest of the accelerator run.

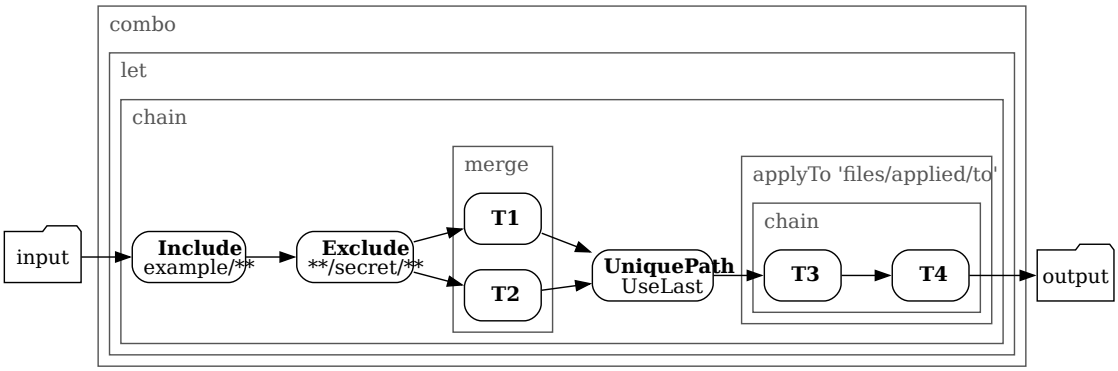
See also

- Conflict Resolution

Combo transform

This topic tells you about the Application Accelerator `Combo` transform in Tanzu Application Platform (commonly known as TAP).

The `Combo` transform combines the behaviors of Include, Exclude, Merge, Chain, UniquePath, and Let.



Syntax reference

Here is the full syntax of `Combo`:

```
type: Combo                  # This can be omitted, because Combo is the default trans
form type.
let:                         # See Let.
  - name: <string>
    expression: <SpEL expression>
  - name: <string>
    expression: <SpEL expression>
condition: <SpEL expression>
include: [<ant pattern>]     # See Include.
exclude: [<ant pattern>]     # See Exclude.
merge:                       # See Merge.
  - <m1-transform>
  - <m2-transform>
  - ...
chain:                      # See Chain.
  - <c1-transform>
  - <c2-transform>
  - ...
applyTo: [<ant pattern>]    # See Chain
onConflict: <conflict resolution> # See UniquePath.
```

Behavior

The `Combo` transform properties have default values, are optional, and you must use at least one property.

When you configure the `Combo` transform with all properties, it behaves as follows:

1. Applies the `include` as if it were the first element of a Chain. The default value is `['**']`; if not present, all files are retained.
2. Applies the `exclude` as if it were the second element of the chain. The default value is `[]`; if not present, no files are excluded. Only files that match the `include`, but are not excluded by the `exclude`, remain.
3. Feeds all those files as input to all transforms declared in the `merge` property, exactly as Merge does. The result of that `Merge`, which is the third transform in the big chain, is another set of files. If there are no elements in `merge`, the previous result is directly fed to the next step.
4. The result of the merge step is prone to generate duplicate entries for the same `path`. It's implicitly forwarded to a UniquePath check, configured with the `onConflict` strategy. The

default policy is to retain files appearing later. The results of the transforms that appear later in the `merge` block "win" against results appearing earlier.

5. Passes that result as the input to the chain defined by the `chain` property. The combo chain is prolonged with the elements defined in `chain`. If there are no elements in `chain`, it's as if the previous result was used directly. If the `applyTo` property is set, it applies to the sub-chain (and that sub-chain only).
6. If the `let` property is defined in the `Combo`, the whole execution is wrapped inside a Let that exposes its derived symbols.

To recap in pseudo code, a giant `Combo` behaves like this:

```
Let(symbols, in:
    Chain(
        include,
        exclude,
        Chain(Merge(<m1-transform>, <m2-transform>, ...), UniquePath(onConflict)),
        Chain(<applyTo>, <c1-transform>, <c2-transform>, ...)
    )
)
```

You rarely use at any one time all the features that `Combo` offers. Yet `Combo` is a good way to author other common building blocks without having to write their `type: x` in full.

For example, this:

```
include: ['**/*.txt']
```

is a perfectly valid way to achieve the same effect as this:

```
type: Include
patterns: ['**/*.txt']
```

Similarly, this:

```
chain:
  - type: T1
    ...
  - type: T2
    ...
```

is often preferred over the more verbose:

```
type: Chain
transformations:
  - type: T1
    ...
  - type: T2
    ...
```

As with other transforms, the order of declaration of properties has no impact. For clarity, a convention that mimics the actual behavior is used, but the following applies **T1** and **T2** on all `.yaml` files even though it places the `include` section after the `merge` section.

```
merge:
  - type: T1
  - type: T2
include: ["*.yaml"]
```

In other words, `Combo` applies `include` filters before `merge` irrespective of the physical order of the keys in the YAML text. It's a good practice to place the `include` key before the `merge` key. This makes the accelerator definition more readable, but has no effect on its execution order.

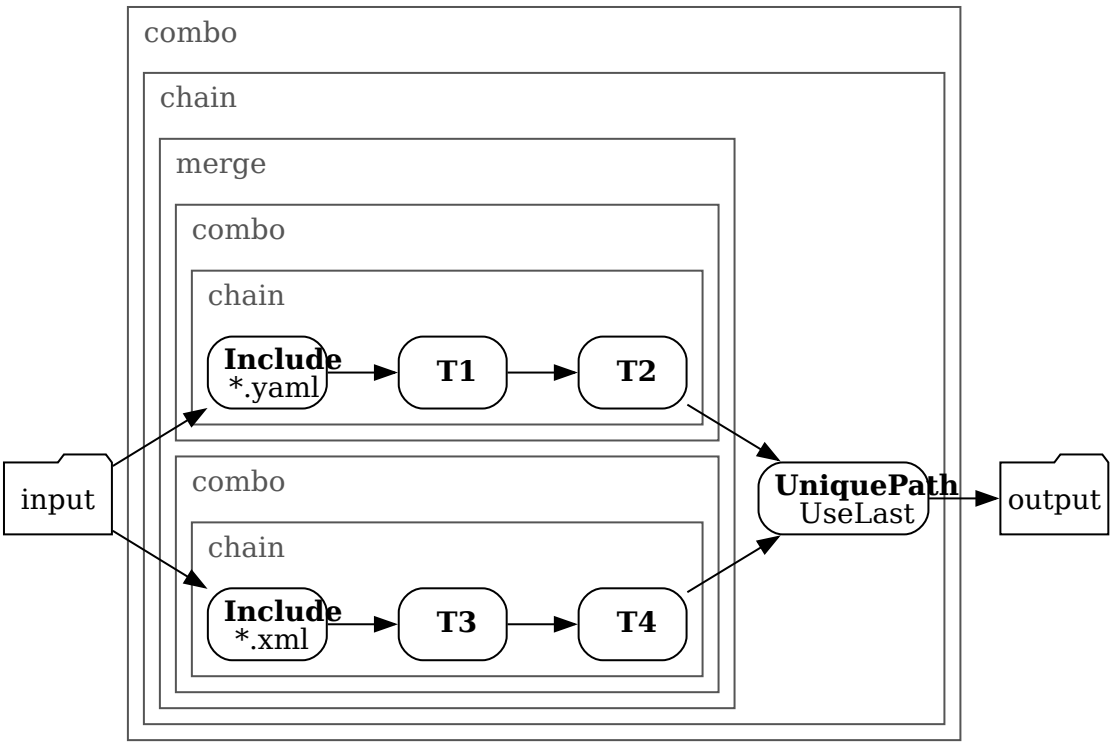
Examples

The following are typical use cases for `Combo`.

Example 1

To apply separate transformations to separate sets of files. For example, to all `.yaml` files and to all `.xml` files:

```
merge:                    # This uses the Merge syntax in a first Combo.
  - include: ['*.yaml']      # This actually nests a second Combo inside the first.
    chain:
      - type: T1
      - type: T2
  - include: ['*.xml']      # Here comes a third Combo, used as the 2nd child inside t
he first
    chain:
      - type: T3
      - type: T4
```

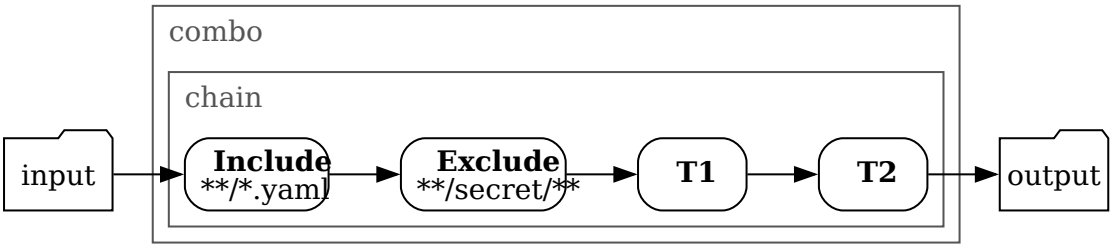


Example 2

To apply **T1** then **T2** on all `.yaml` files that are not in any `secret` directory:

```
include: ['**/*.yaml']
exclude: ['**/secret/**']
chain:
  - type: T1
    ..
```

```
  - type: T2
    ..
```



Include transform

This topic tells you about the Application Accelerator `Include` transform in Tanzu Application Platform (commonly known as TAP).

The `Include` transform retains files based on their `path`, letting in *only* those files whose path matches at least one of the configured `patterns`. The contents of files, and any of their other characteristics, are unaffected.

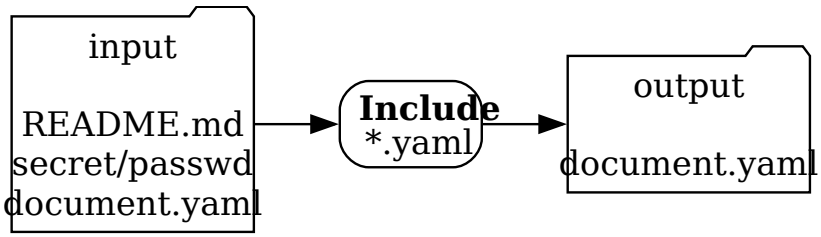
`Include` is a basic building block seldom used as is, which makes sense if composed inside a Chain or a Merge. It is often more convenient to leverage the shorthand notation offered by Combo.

Syntax reference

```
type: Include
patterns: [<ant pattern>]
condition: <SpEL expression>
```

Examples

```
type: Chain
transformations:
  - type: Include
    patterns: ["**/*.yaml"]
  - type: # At this point, only yaml files are affected
```



See also

- Exclude
- Combo

Exclude transform

This topic tells you about the Application Accelerator `Exclude` transform in Tanzu Application Platform (commonly known as TAP).

The `Exclude` transform retains files based on their `path`, allowing all files except ones with a path that matches at least one of the configured `patterns`. The contents of files, and any of their other characteristics are unaffected.

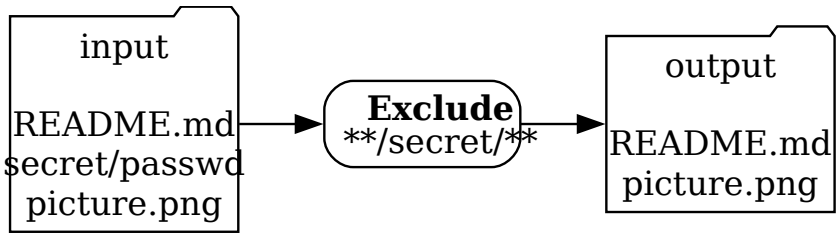
`Exclude` is a basic building block seldom used *as is*, which makes sense if composed inside a Chain or a Merge. It is often more convenient to leverage the shorthand notation offered by Combo.

Syntax reference

```
type: Exclude
patterns: [<ant pattern>]
condition: <SpEL expression>
```

Examples

```
type: Chain
transformations:
  - type: Exclude
    patterns: ["**/secret/**"]
  - type: # At this point, no file matching **/secret/** is affected.
```



See also

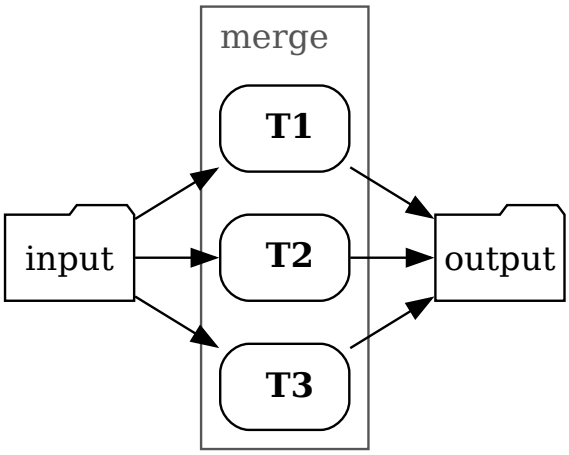
- Include
- Combo

Merge transform

This topic tells you about the Application Accelerator `Merge` transform in Tanzu Application Platform (commonly known as TAP).

The `Merge` transform feeds a copy of its input to several other transforms and merges the results together using set union.

A `Merge` of **T1**, **T2**, and **T3** applied to input **I** results in **T1(I)** ∪ **T2(I)** ∪ **T3(I)**.



An empty merge produces nothing (∅).

Syntax reference

```
type: Merge
sources:
  - <transform>
  - <transform>
  - <transform>
  - ...
condition: <SpEL expression>
```

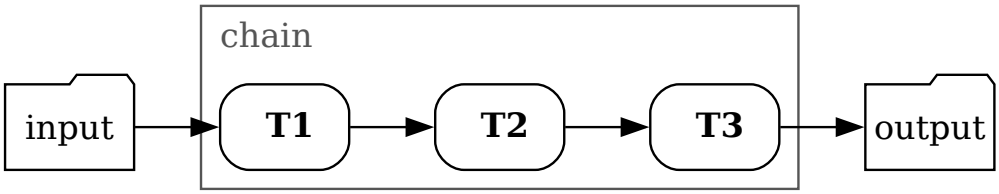
See also

- Combo is often used to express a `Merge` **and** other transformations in a shorthand syntax.

Chain transform

This topic tells you about the Application Accelerator `Chain` transform in Tanzu Application Platform (commonly known as TAP).

The `Chain` transform uses function composition to produce its final output.



Syntax reference

```
type: Chain
transformations:
  - <transform>
  - <transform>
  - <transform>
  - ...
applyTo: [<ant pattern>]
condition: <SpEL expression>
```

Behavior

A chain of **T1** then **T2** then **T3** first applies transform **T1**. It then applies **T2** to the output of **T1**, and finally applies **T3** to the output of that. In other words, **T3** to **T2** to **T1**.

An empty chain acts as function identity.

If the optional `applyTo` property is set, then the chained transformations are only applied to files with paths that match the `applyTo` patterns. Files with paths that don't match are left untouched and merged back with the other results to form the final result of the `Chain` transform.

Let transform

This topic tells you about the Application Accelerator `Let` transform in Tanzu Application Platform (commonly known as TAP).

The `Let` transform wraps another transform, creating a new scope that extends the existing scope.

SpEL expressions inside the `Let` can access variables from both the existing scope and the new scope.

Variables defined by the `Let` should not shadow existing variables. If they do, those existing variables won't be accessible.

Syntax reference

```
type: Let
symbols:
- name: <string>
  expression: <SpEL expression>
- ...
in: <transform> # <- new symbols are visible in here
```

Execution

The `Let` adds variables to the new scope by computation of SpEL expressions.

```
engine:
  let:
  - name: <string>
    expression: <SpEL expression>
  - ...
```

Both a `name` and an `expression` must define each symbol where:

- `name` must be a camelCase string name. If a let *symbol* happens to have the same name as a symbol already defined in the surrounding scope, then the local symbol shadows the symbol from the surrounding scope. This makes the variable from the surrounding scope inaccessible in the remainder of the `Let` but doesn't alter its original value.
- `expression` must be a valid SpEL expression expressed as a YAML string. Be careful when using the `#` symbol for variable evaluation, because this is the comment marker in YAML. So SpEL expressions in YAML must enclose strings in quotes or rely on block style. For more information about block style, see Block Style Productions.

Symbols defined in the `Let` are evaluated in the new scope in the order they are defined. This means that symbols lower in the list can make use of the variables defined higher in the list but not the other way around.

See also

- Combo provides a way to declare a `Let` scope and other transforms in a short syntax.

Loop transform

This topic tells you about the Application Accelerator `Loop` transform in Tanzu Application Platform (commonly known as TAP).

The `Loop` transform iterates over elements in a list and applies the provided transform for every element in that list.

When `doAsMerge` is used, a copy of the `Loop` transform's input is passed to each transform and the outputs from each transform are merged using a set union.

When `doAsChain` is used, each transform is executed sequentially, receiving the previous transform's output as its input. The first transform is to receive the `Loop` transform's input as its input.

Syntax reference

```
type: Loop
on: <SpEL expression>
var: <string>
index: <string>
doAsChain: <transform>
doAsMerge: <transform>
```

- `on` must be a SpEL expression that evaluates a list. This is the list of elements to be iterated over.
- `var` is the name of the variable to be assigned to the current element on each iteration. (optional)
- `index` is the variable's name to be assigned to the index of the current element on each iteration. (optional)
- `doAsMerge` is the transform to be executed for every element in the list, on a copy of the `Loop` transform's input.
- `doAsChain` is the transform to be executed for every element in the list, passing the output of the transform as input to the next transform.

Both `var` and `index` are optional.

Only one of the `doAsMerge` or `doAsChain` variables is to be used in a `Loop` transform.

Behavior

Consider the following when choosing `doAsMerge` or `doAsChain`:

`doAsMerge` executes the transform on the same input files for every iteration and merges the resulting outputs. It is best suited when a transform is executed multiple times on the same input and does not have conflicts.

`doAsChain` executes the transform on the initial input files once and then passes the resulting output to the second iteration and so on. It is best suited when a transform must detect any changes that occurred in the previous iteration.

Examples

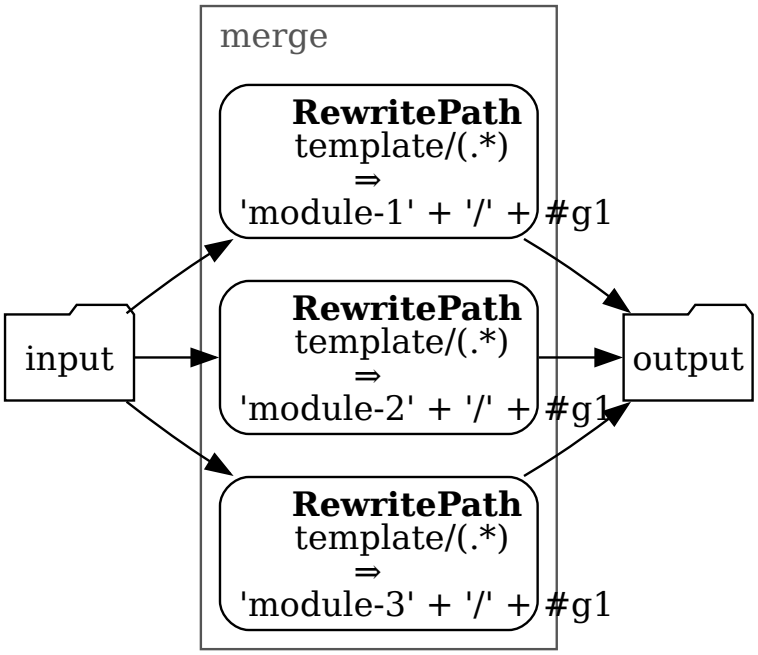
See the following examples using the `Loop` transform.

Example 1

Create a new directory for every module in `modules` (a list of strings) based on the contents of the "template" directory.

```
type: Loop
on: "#modules"
var: m
doAsMerge:
  type: RewritePath
  regex: "template/(.*)"
  rewriteTo: "#m + '/' + #g1"
```

The following diagram shows how this example behaves:

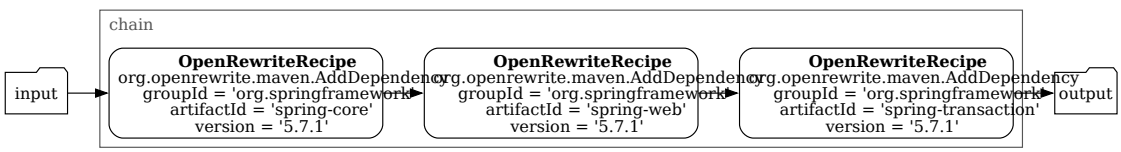


Example 2

Add every artifactId in `artifacts` (a list of strings) as a Spring dependency.

```
type: Loop
on: "#artifacts"
var: a
doAsChain:
  type: OpenRewriteRecipe
  recipe: org.openrewrite.maven.AddDependency
  options:
    groupId: "'org.springframework'"
    artifactId: "#a"
    version: "'5.7.1'"
```

The following diagram shows how this example behaves:



Example 3

You can use `Loop` in combination with custom types, for example:

```
accelerator:
  types:
    - name: MavenPlugin
      struct:
        - name: groupId
        - name: artifactId
        - name: version
  options:
    - name: pluginsToAdd
      dataType: [MavenPlugin] # End users will be able to enter a collection of GAV tu
ples
engine:
  include: [pom.xml]
  chain:
    - type: Loop
      on: pluginsToAdd # Iterate on the pluginsToAdd collection
      var: p           # The variable "p" will contain each tuple in turn
      doAsChain:       # Will apply the second execution to the result of the first, a
nd so on...
        type: OpenRewriteRecipe
        recipe: org.openrewrite.maven.AddPlugin
        options:
          groupId:    "#p['groupId']"
          artifactId: "#p['artifactId']"
          version:    "#p['version']"
```

For more information, see Using Custom Types.

InvokeFragment transform

This topic tells you about the Application Accelerator `InvokeFragment` transform in Tanzu Application Platform (commonly known as TAP).

The `InvokeFragment` performs transformations defined in an imported Fragment, allowing re-use across accelerators.

Syntax reference

```
type: InvokeFragment
reference: <imported-fragment>
let:  # See Let
  - name: <string>
    expression: <SpEL expression>
  ...
anchor: [<file path>]
```

Behavior

Assuming some fragment `my-fragment` has been imported in the accelerator (thus exposing the options it defines as options of the current accelerator), the following construct invokes `my-fragment`:

```
type: InvokeFragment
reference: my-fragment
```

This passes all input files (depending where this invocation sits in the “tree”) to the invoked fragment, which can then manipulate them alongside its own files. The result of the invocation becomes the result of this transform.

Variables

At the point of invocation, all currently defined variables are made visible to the invoked fragment. Therefore, if it was `import`-ed in the most straightforward manner, a fragment defining an option `myOption` is defining an option named `myOption` at the accelerator level, and the value provided by the user is visible at the time of invocation.

To override a value, or if an imported option has been exposed under a different name, or not at all, you can use a `let` construct when using `InvokeFragment`. This behaves as the Let transform: for the duration of the fragment invocation, the variables defined by `let` now have their newly defined values. Outside the scope of the invocation, the regular model applies.

Files

The set of files coming from the invoking accelerator and made visible to the fragment is the set of files that “reach” the point of invocation. For example, in the following case:

```
include: ["somedir/**"]
chain:
  - type: InvokeFragment
    reference: my-fragment
```

All files that the fragment invocation “sees” are files in the `somedir/` subdirectory. If the `my-fragment` has not been written accordingly, this can be problematic. Chances are that this re-usable fragment expects files to be present at the root of the project tree and work on them.

To better cope with this typical situation, the `InvokeFragment` transform exposes the optional `anchor` configuration property. Continuing with the earlier example, by using `anchor: somedir`, then all files coming from the current accelerator are exposed as if their `path` had the `somedir/` prefix removed. When it comes to gathering the result of the invocation though, all resulting files are re-introduced with a prefix prepended to their `path` (this applies to **all** files produced by the fragment, not just the ones originating from the accelerator).

The value of the `anchor` property must not start nor end with a slash (`/`) character.

Examples

The following is a full-featured example showcasing the interaction between the `imports` section and `InvokeFragment`:

```
accelerator:
  name: my-accelerator
  options:
    - name: someOption
      dataType: number
  imports:
    - name: my-fragment

engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
        - include: ["**/pom.xml"]
```

```
        - type: InvokeFragment
          reference: my-fragment
```

Assuming `my-fragment` is defined as follows:

```
accelerator:
  name: my-fragment
  options:
    - name: indentationLevel
      dataType: number
      defaultValue: 2
transform:
  chain:
    - include: ["**/*.xml"]
    - type: SomeTransform
      ...
```

Then users will be presented with two options: `someOption` and `indentationLevel`, as if `indentationLevel` was defined in the host accelerator.

Moreover, the behavior of the calling accelerator is exactly as if the body of the fragment transform was inserted in-place of `InvokeFragment`:

```
accelerator:
  name: my-accelerator
  options:
    - name: someOption
      dataType: number
    - name: indentationLevel
      dataType: number
      defaultValue: 2

engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
        - include: ["**/pom.xml"]
        - chain:
          - include: ["**/*.xml"]
          - type: SomeTransform
            ...
```

Now you can imagine some scenarios to better clarify all configuration properties.

If, for some reason, you don't want to use the value entered in the `indentationLevel` option for the fragment, but twice the value provided for `someOption`. The `InvokeFragment` block can be rewritten as follows:

```
    type: InvokeFragment
    reference: my-fragment
    let:
      - name: indentationLevel
        value: '2 * #someOption'
```

Finally, if the invocation in the accelerator looks like this:

```
engine:
  merge:
    - include: ["..."]
    - ...
    - chain:
```

```
        - include: ["**/README.md"]
        - type: InvokeFragment
          reference: my-fragment
```

Then there is zero visible effect, because this is forwarding only `README.md` files to the fragment and the fragment is itself using a filter on `*.xml` files.

See also

- Let
- RewritePath

ReplaceText transform

This topic tells you about the Application Accelerator `ReplaceText` transform in Tanzu Application Platform (commonly known as TAP).

The `ReplaceText` transform allows replacing one or several text tokens in files as they are being copied to their destination. The replacement values are the result of dynamic evaluation of SpEL expressions.

This transform is text-oriented and requires knowledge of how to interpret the stream of bytes that make up the file contents into text. All files are assumed to use `UTF-8` encoding by default, but you can use the UseEncoding transform upfront to specify a different charset to use on some files.

You can use `ReplaceText` transform in one of two ways:

- To replace several literal text tokens.
- To define the replacement behavior using a single regular expression, in which case the replacement SpEL expression can leverage the regex capturing group syntax.

Syntax reference

Syntax reference for replacing several literal text tokens:

```
type: ReplaceText
substitutions:
  - text: STRING
    with: SPEL-EXPRESSION
  - text: STRING
    with: SPEL-EXPRESSION
  - ..
condition: SPEL-EXPRESSION
```

Syntax reference for defining the replacement behavior using a *single* regular expression:

Regex is used to match the entire document. To match on a per line basis, enable multiline mode by including `(?m)` in the regex.

```
type: ReplaceText
regex:
  pattern: REGULAR-EXPRESSION
  with: SPEL-EXPRESSION
condition: SPEL-EXPRESSION
```

In both cases, the SpEL expression can use the special `#files` helper object. This enables the replacement string to consist of the contents of an accelerator file. See the following example.

Another set of helper objects are functions of the form `xxx2Yyyy()` where `xxx` and `yyy` can take the value `camel`, `kebab`, `pascal`, or `snake`. For example, `camel2Snake()` enables changing from camelCase to snake_case.

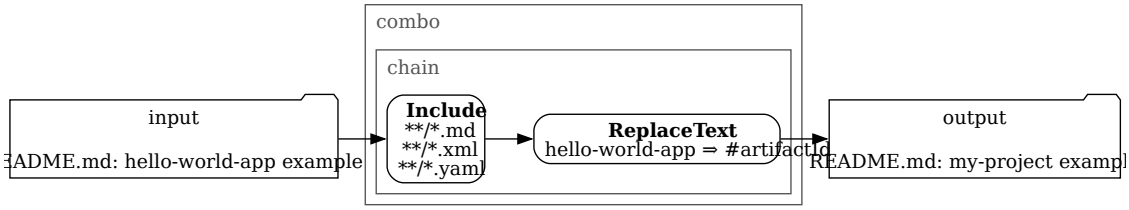
Examples

See the following examples using The `ReplaceText` transform.

Example 1

Replacing the hardcoded string `"hello-world-app"` with the value of variable `#artifactId` in all `.md`, `.xml`, and `.yaml` files.

```
include: ['**/*.md', '**/*.xml', '**/*.yaml']
chain:
  - type: ReplaceText
    substitutions:
      - text: "hello-world-app"
        with: "#artifactId"
```



Example 2

Replacing the hardcoded string `"hello-world-app"` with the value of variable `#artifactId` in the `README-fr.md` and `README-de.md` files, which are encoded using the `ISO-8859-1` charset:

```
include: ['README-fr.md', 'README-de.md']
chain:
  - type: UseEncoding
    encoding: 'ISO-8859-1'
  - type: ReplaceText
    substitutions:
      - text: "hello-world-app"
        with: "#artifactId"
```

Example 3

Similar to the preceding example, but making sure the value appears as kebab case, while the entered `#artifactId` is using camel case:

```
include: ['**/*.md', '**/*.xml', '**/*.yaml']
chain:
  - type: ReplaceText
    substitutions:
      - text: "hello-world-app"
        with: "#camel2Kebab(#artifactId)"
```

Example 4

Replacing the hardcoded string `"REPLACE-ME"` with the contents of file named after the value of the `#platform` option in `README.md`:

```
include: ['README.md']
chain:
  - type: ReplaceText
    substitutions:
      - text: "REPLACE-ME"
        with: "#files.contentsOf('snippets/install-' + #platform + '.md')"
```

See also

- UseEncoding

RewritePath transform

This topic tells you about the Application Accelerator `RewritePath` transform in Tanzu Application Platform (commonly known as TAP).

The `RewritePath` transform allows you to change the name and path of files without affecting their content.

Syntax reference

```
type: RewritePath
regex: <string>
rewriteTo: <SpEL expression>
matchOrFail: <boolean>
```

For each input file, `RewritePath` attempts to match its `path` by using the regular expression (regex) defined by the `regex` property. If the regex matches, `RewritePath` changes the `path` of the file to the evaluation result of `rewriteTo`.

`rewriteTo` is an expression that has access to the overall engine model and to variables defined by capturing groups of the regular expression. Both *named capturing groups* `(?<example>[a-z]*)` and regular *index-based* capturing groups are supported. `g0` contains the whole match, `g1` contains the first capturing group, and so on.

If the regex doesn't match, the behavior depends on the `matchOrFail` property:

- If set to `false`, which is the default, the file is left untouched.
- If set to `true`, an error occurs. This prevents misconfiguration if you expect all files coming in to match the regex. For more information about typical interactions between `RewritePath` and `Chain + Include`, see the following section, Interaction with Chain and Include.

The default value for `regex` is the following regular expression, which provides convenient access to some named capturing groups:

```
^(?<folder>.*/)?(?<filename>([^/]+?|)(?=(?<ext>\.[^/.]*)?)$)
```

Using `some/deep/nested/file.xml` as an example, the preceding regular expression captures:

- **folder:** The full folder path the file is in. In this example, `some/deep/nested/`.
- **filename:** The full name of the file, including extension *if present*. In this example, `file.xml`.
- **ext:** The last dot and extension in the filename, *if present*. In this example, `.xml`.

The default value for `rewriteTo` is the expression `#folder + #filename`, which doesn't rewrite paths.

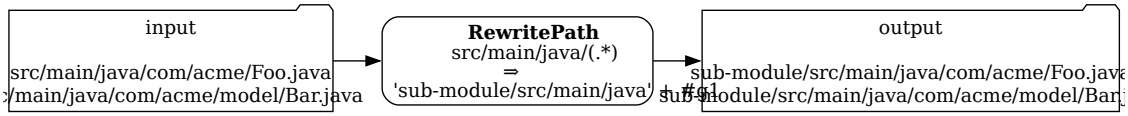
Examples

See the following examples using the `RewritePath` transform.

Example 1

The following moves all files from `src/main/java` to `sub-module/src/main/java`:

```
type: RewritePath
regex: src/main/java/(.*)
rewriteTo: "'sub-module/src/main/java' + #g1"   # 'sub-module/' + #g0 works too
```



Example 2

The following flattens all files found inside the `sub-path` directory and its subdirectories, and puts them into the `flattened` folder:

```
type: RewritePath
regex: sub-path/(.*/)*(?<filename>[^/]+)
rewriteTo: "'flattened' + #filename"   # 'flattened' + #g2 would work too
```

Example 3

The following turns all paths into lowercase:

```
type: RewritePath
rewriteTo: "#g0.toLowerCase()"
```

Interaction with Chain and Include

It's common to define pipelines that perform a `Chain` of transformations on a subset of files, typically selected by `Include/Exclude`:

```
- include: ["**/*.java"]
- chain:
    - # do something here
    - # and then here
```

If one of the transformations in the chain is a `RewritePath` operation, chances are you want the rewrite to apply to *all* files matched by the `Include`. For those typical configurations, you can set the `matchOrFail` guard to `true` to ensure the `regex` you provide indeed matches all files coming in.

See also

- Use UniquePath to ensure rewritten paths don't clash with other files, or to decide which path to select if they do clash.

OpenRewriteRecipe transform

This topic tells you about the Application Accelerator `OpenRewriteRecipe` transform in Tanzu Application Platform (commonly known as TAP).

The `OpenRewriteRecipe` transform allows you to apply any Open Rewrite **Recipe** to a set of files and gather the results.

The following Open Rewrite Recipes are supported:

- Java recipes
- Maven recipes
- XML recipes
- YAML recipes
- JSON recipes
- Properties recipes

The engine leverages v7.30.1 of Open Rewrite and parses Java files using the grammar for Java 11.

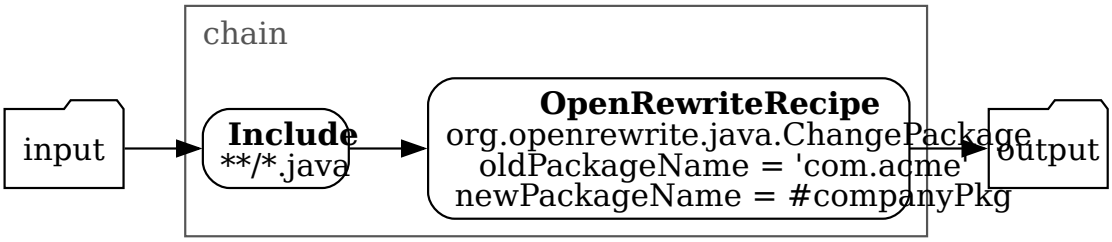
Syntax reference

```
type: OpenRewriteRecipe
recipe: <string>                    # Full qualified classname of the recipe
options:
  <string>: <SpEL expression>       # Keys and values depend on the class of the recipe
  <string>: <SpEL expression>       # Refer to the documentation of said recipe
  ...
```

Example

The following example applies the ChangePackage Recipe to a set of Java files in the `com.acme` package and moves them to the value of `#companyPkg`. This is more powerful than using RewritePath and ReplaceText, as it reads the syntax of files and correctly deals with imports, fully compared to non-fully qualified names, and so on.

```
chain:
  - include: ["**/*.java"]
  - type: OpenRewriteRecipe
    recipe: org.openrewrite.java.ChangePackage
    options:
      oldPackageName: "'com.acme'"
      newPackageName: "#companyPkg"
```



YTT transform

This topic tells you about the Application Accelerator `YTT` transform in Tanzu Application Platform (commonly known as TAP).

The `YTT` transform starts the YTT template engine as an external process.

Syntax reference

```
type: YTT
extraArgs: # optional
  - <SPEL-EXPRESSION-1>
  - <SPEL-EXPRESSION-2>
  - ...
```

The `YTT` transform's YAML notation does not require any parameters. When invoked without parameters, which is the typical use case, the YTT transform's input is determined entirely by two things only:

1. The input files fed into the transform.
2. The current values for options and derived symbols.

Execution

YTT is invoked as an external process with the following command line:

```
ytt -f <input-folder> \
    --data-values-file <symbols.json> \
    --output-files <output-folder> \
    <extra-args>
```

The `<input-folder>` is a temporary directory into which the input files are "materialized." That is, the set of files passed to the YTT transform as input is written out into this directory to allow the YTT process to read them.

The `<symbols.json>` file is a temporary JSON file, which the current option values and derived symbols are materialized in the form of a JSON map. This allows YTT templates in the `<input-folder>` to make use of these symbols during processing.

The `<output-folder>` is a fresh temporary directory that is empty at the time of invocation. In a typical scenario, upon completion, the output directory contains files generated by YTT.

The `<extra-args>` are additional command line arguments obtained by evaluating the SPEL expressions from the `extraArgs` attribute.

When the `ytt` process completes with a 0 exit code, this is considered a successful execution and the contents of the output directory is taken to be the result of the YTT transform.

When the `ytt` process completes with a non 0 exit code, the execution of the `YTT` transform is considered to have failed and an exception is raised.

Examples

See the following examples using the `YTT` transform.

Basic invocation

When you want to execute `ytt` on the contents of the entire accelerator repository, use the YTT transform as your only transform in the engine declaration.

```
accelerator:
  ...
engine:
  type: YTT
```

To do anything beyond calling YTT, compose YTT into your accelerator flow using merge or chain combinators. This is exactly the same as composing any other type of transform.

For example, when you want to define some derived symbols as well as merge the results from YTT with results from other parts of your accelerator transform, you can reference this example:

```
engine:
  let: # Define derived symbols visible to all transforms (including YTT)
  - name: theAnswer
    expression: "41 + 1"
  merge:
  - include: ["deploy/**.yml"] # select some yaml files to process with YTT
    chain: # Chain selected yaml files to YTT
    - type: YTT
  - ... include/generate other stuff to be merged alongside yaml generated by YTT...
```

The preceding example uses a combination of Chain and Merge. You can use either `Merge` or `Chain` or both to compose YTT into your accelerator flow. Which one you choose depends on how you want to use YTT as part of your larger accelerator.

Using `extraArgs`

The `extraArgs` passes additional command line arguments to YTT. This adds file marks. See File Marks in the Carvel documentation.

For example, the following runs YTT and renames the `foo/demo.yml` file in its output to `bar/demo.yml`.

```
engine:
  type: YTT
  extraArgs: ["'--file-mark'",  "'foo/demo.yml:path=bar/demo.yml'"]
```

The `extraArgs` attribute expects SPEL expressions. Take care to use proper escaping of literal strings using double and single quotes (that is, `“‘LITERAL-STRING’”).

UseEncoding transform

This topic tells you about the Application Accelerator `UseEncoding` transform in Tanzu Application Platform (commonly known as TAP).

When considering files in textual form, for example, when doing text replacement with the ReplaceText transform, the engine must decide which encoding to use.

By default, `UTF-8` is assumed. If any files must be handled differently, use the `UseEncoding` transform to annotate them with an explicit encoding.

`UseEncoding` returns an error if you apply encoding to files that have already been explicitly configured with a particular encoding.

Syntax reference

```
type: UseEncoding
encoding: <encoding>    # As recognized by the java java.nio.charset.Charset class
condition: <SpEL expression>
```

Supported encoding names include, for example, `UTF-8`, `US-ASCII`, and `ISO-8859-1`.

Example use

`UseEncoding` is typically used as an upfront transform to, for example, ReplaceText in a chain:

```
type: Chain   # Or using "Combo"
transformations:
  - type: UseEncoding
    encoding: ISO-8859-1
  - type: ReplaceText
    substitutions:
      - text: "hello"
        with: "#howToSayHello"
```

See also

- ReplaceText

UniquePath transform

This topic tells you about the Application Accelerator `UniquePath` transform in Tanzu Application Platform (commonly known as TAP).

You can use the `UniquePath` transform to ensure there are no `path` conflicts between files transformed. You can often use this at the tail of a Chain.

Syntax reference

```
type: UniquePath
strategy: <conflict resolution>
condition: <SpEL expression>
```

Examples

The following example concatenates the file that was originally named `DEPLOYMENT.md` to the file `README.md`:

```
chain:
  - merge:
      - include: ['README.md']
      - include: ['DEPLOYMENT.md']
        chain:
          - type: RewritePath
            rewriteTo: "'README.md'"
  - type: UniquePath
    strategy: Append
```

See also

- `UniquePath` uses a Conflict Resolution strategy to decide what to do when several input files use the same `path`.
- Combo implicitly embeds a `UniquePath` after the Merge defined by its `merge` property.

Conflict resolution

This topic tells you how to resolve conflicts that Application Accelerator transforms in Tanzu Application Platform (commonly known as TAP) might produce.

For example, if you're using Merge (or Combo's `merge` syntax) or RewritePath, a transform can produce several files at the same `path`. The engine then must take an action: Should it keep the last

file? Report an error? Concatenate the files together?

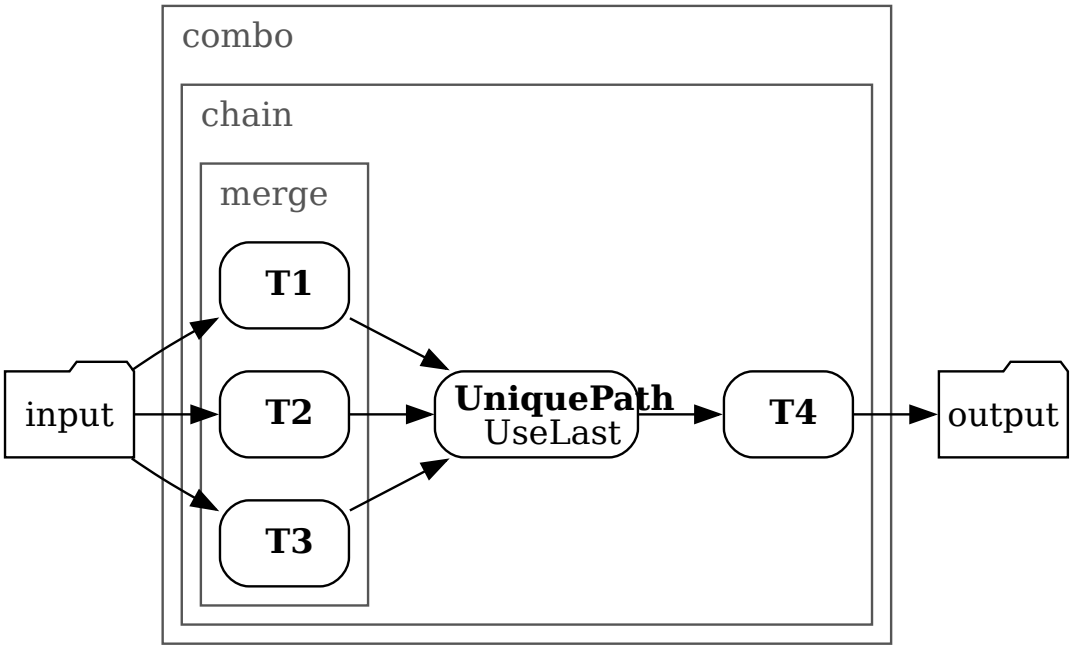
Syntax reference

Conflicts can arise for a number of reasons. You can avoid or resolve them by configuring transforms with a conflict resolution. For example:

- Combo uses UseLast by default, but you can configure it to do otherwise.
- You can explicitly end a transform Chain with a UniquePath, which by default uses Fail. This is customizable.

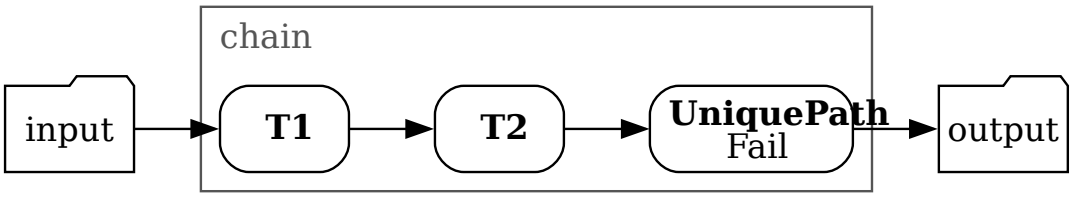
Combo

```
type: Combo      # often omitted
merge:
  - <transform>
  - <transform>
  - <transform>
chain:
  - <transform>
  - ...
onConflict: <conflict resolution>  # defaults to 'UseLast'
```



Chain

```
type: Chain      # or implicitly using Combo
transformations:
  - <transform>
  - <transform>
  - type: UniquePath
    strategy: <conflict resolution>  # defaults to 'Fail'
```



Available strategies

The following values and behaviors are available:

- `Fail`: Stop processing on the first file that exhibits `path` conflicts.
- `UseFirst`: For each conflicting file, the file produced first (typically by a transform appearing earlier in the YAML definition) is retained.
- `UseLast`: For each conflicting file, the file produced last (typically by a transform appearing later in the YAML definition) is retained.
- `Append`: The conflicting versions of files are concatenated (as if using `cat file1 file2 ...`), with files produced first appearing first.
- `FavorOwn`: Only makes sense in the context of composition. Selects the version of the file that comes from the current executing fragment if possible, falls back to the caller version otherwise.
- `FavorForeign`: Only makes sense in the context of composition. Selects the version of the file that was provided by the caller if present, falls back to the file originating from this fragment's fileset otherwise.
- `NWayDiff`: Try to merge the conflicting resources by applying patches computed against a common ancestor. The resulting resource has the attributes of the first conflicting resource.

See also

- Combo
- UniquePath

Provenance transform

This topic tells you about the Application Accelerator `Provenance` transform in Tanzu Application Platform (commonly known as TAP).

The `Provenance` transform is a special transform used to generate a file that provides details of the accelerator engine transform.

For more information, see Application Bootstrapping Provenance.

Syntax reference

```
type: Provenance
condition: <SpEL expression>
```

The `Provenance` transform is added as a child to the top-most transform, which is usually a `Merge` or a `Chain`, using a `Combo`.

Behavior

The `Provenance` transform ignores its input and outputs a single resource named `accelerator-info.yaml`. For example:

```
id: <unique GUID of invocation>
timestamp: <timestamp in RFC3339 format>
username: <captured username of user triggering the run>
source: <client environment from which accelerator was run>
accelerator:
  name: <name of registered accelerator>
  git:
    url: <git repository location>
    ref:
      branch: <branch name> or
      tag: <tag name> or
      commit: <specific requested commit>
    subPath: <optional subpath inside the repo>
    commit: <actual SHA the branch or tag pointed to>
fragments:
  - name: <name of registered fragment 1>
    git:
      url: <git repository location>
      ref:
        branch: <branch name> or
        tag: <tag name> or
        commit: <specific requested commit>
      subPath: <optional subpath inside the repo>
      commit: <actual SHA the branch or tag pointed to>
  - name: <name of registered fragment 2>
    git:
      url: <git repository location>
      ref:
        branch: <branch name> or
        tag: <tag name> or
        commit: <specific requested commit>
      subPath: <optional subpath inside the repo>
      commit: <actual SHA the branch or tag pointed to>
  - ...
options:
  - name: <option name>
    value: <option value>
  - name: <option name>
    value: <option value>
```

Note

Depending on the invocation scenario, some pieces of data might not be present.

Use SpEL with Application Accelerator

This topic tells you about some common Spring Expression Language (SpEL) use cases for the `accelerator.yaml` file in Application Accelerator.

For more information, see Spring Expression Language documentation.

Variables

You can reference all the values added as options in the `accelerator` section from the YAML file as variables in the `engine` section. You can access the value using the syntax `#<option name>`:

```
options:
  - name: foo
    dataType: string
    inputType: text
...
engine:
  - include: ["some/file.txt"]
    chain:
    - type: ReplaceText
      substitutions:
      - text: bar
        with: "#foo"
```

This sample replaces every occurrence of the text `bar` in the file `some/file.txt` with the contents of the `foo` option.

Implicit variables

Some variables are made available to the model by the engine, including:

- `artifactId` is a built-in value derived from the `projectName` passed in from the UI with spaces replaced by "_". If that value is empty, it is set to `app`.
- `files` is a helper object that currently exposes the `contentsOf(<path>)` method. For more information, see ReplaceText.
- `camel2Kebab` and other variations of the form `xxx2Yyyy` is a series of helper functions for dealing with changing case of words. For more information, see ReplaceText.

Conditionals

You can use Boolean options for conditionals in your transformations.

```
options:
  - name: numbers
    inputType: select
    choices:
    - text: First Option
      value: first
    - text: Seconf Option
      value: second
    defaultValue: first
...
engine:
  - include: ["some/file.txt"]
    condition: "#numbers == 'first'"
    chain:
    - type: ReplaceText
      substitutions:
      - text: bar
        with: "#foo"
```

This replaces the text only if the selected option is the first one.

Rewrite path concatenation

```
options:
  - name: renameTo
    dataType: string
    inputType: text
...
```

```
engine:
  - include: ["some/file.txt"]
    chain:
    - type: RewritePath
      rewriteTo: "'somewhere/' + #renameTo + '.txt'"
```

Regular expressions

Regular expressions allow you to use patterns as a matcher for strings. Here is a small example of what you can do with them:

```
options:
  - name: foo
    dataType: string
    inputType: text
    defaultValue: abcZ123
...
engine:
  - include: ["some/file.txt"]
    condition: "#foo.matches('[a-z]+Z\d+')"
    chain:
    - type: ReplaceText
      substitutions:
      - text: bar
        with: "#foo"
```

This example uses RegEx to match a string of letters that ends with a capital Z and any number of digits. If this condition is fulfilled, the text is replaced in the file, `file.txt`.

Dealing with string arrays

Options with a `dataType` of `[string]` come out as an array of strings.

To use them and for example format the result as a bulleted list, you can use the Java static String.join() method. For example:

```
accelerator:
  options:
    - name: meals
      dataType: [string]
      inputType: checkbox
      choices:
        - value: fish
        - value: chips
        - value: BLT
...
engine:
  type: ReplaceText
  substitutions:
  - text: recipe
    with: "' * ' + T(java.lang.String).join('\n * ', #meals)"
```

Accelerator custom resource definition

This topic tells you about the Application Accelerator custom resource definition.

Overview

The `Accelerator` custom resource definition (CRD) defines any accelerator resources to be made available to the Application Accelerator for VMware Tanzu system. It is a namespaced CRD, meaning that any resources created belong to a namespace. For the resource to be available to the Application Accelerator system, it must be created in the namespace that the Application Accelerator UI server is configured to watch.

The `Fragment` custom resource definition (CRD) defines any accelerator fragment resources to be made available to the Application Accelerator for VMware Tanzu system. It is a namespaced CRD, meaning that any resources created belong to a namespace. For the resource to be available to the Application Accelerator system, it must be created in the namespace that the Application Accelerator UI server is configured to watch.

API definitions

The `Accelerator` CRD is defined with the following properties:

Property	Value
Name	Accelerator
Group	accelerator.apps.tanzu.vmware.com
Version	v1alpha1
ShortName	acc

Accelerator CRD Spec

The `Accelerator` CRD *spec* defined in the `AcceleratorSpec` type has the following fields:

Field	Description	Required/Optional
displayName	A short descriptive name used for an Accelerator.	Optional (*)
description	A longer description of an Accelerator.	Optional (*)
iconUrl	A URL for an image to represent the Accelerator in a UI.	Optional (*)
tags	An array of strings defining attributes of the Accelerator that can be used in a search.	Optional (*)
git	Defines the accelerator source Git repository.	Optional (***)
git.url	The repository URL, can be a HTTP/S or SSH address.	Optional (***)
git.gitImplementation	Determines which git client library to use. The default setting is to go-git. Valid values are (‘go-git’, ‘libgit2’).	Optional (**)
git.ignore	Overrides the set of excluded patterns in the .sourceignore format (which is the same as .gitignore). If not provided, a default of `.git/` is used.	Optional (**)
git.interval	The interval at which to inquire for repository updates. If not provided the default setting is 10 minuets. There is an additional refresh interval (currently 10s) involved before accelerators can appear in the UI. There might be a 10s delay before changes are reflected in the UI.*	Optional (**)
git.ref	Git reference to checkout and monitor for changes, the default is main branch.	Optional (**)
git.ref.branch	The Git branch to checkout, the default is main.	Optional (**)
git.ref.commit	The Git commit SHA to checkout, if specified tag filters are ignored.	Optional (**)

Field	Description	Required/Optional
git.ref.semver	The Git tag semver expression, takes precedence over tag.	Optional (**)
git.ref.tag	The Git tag to checkout, takes precedence over branch.	Optional (**)
git.secretRef	The secret name containing the Git credentials. For HTTPS repositories, the secret must contain user name and password fields. For SSH repositories, the secret must contain identity, identity.pub, and known_hosts fields.	Optional (**)
git.subPath	SubPath is the folder inside the git repository to consider as the root of the accelerator or fragment. Defaults at the root of the repository.	Optional
source	Defines the source image repository.	Optional (***)
source.image	Image is a reference to an image in a remote registry.	Optional (***)
source.imagePullSecrets	ImagePullSecrets contains the names of the Kubernetes Secrets containing registry login information to resolve image metadata	Optional
source.interval	The interval at which to check for repository updates.	Optional
source.serviceAccountName	ServiceAccountName is the name of the Kubernetes ServiceAccount used to authenticate the image pull if the service account has attached pull secrets	Optional

The `Fragment` CRD is defined with the following properties:

Property	Value
Name	Fragment
Group	accelerator.apps.tanzu.vmware.com
Version	v1alpha1
ShortName	frag

Fragment CRD Spec

The `Fragment` CRD *spec* defined in the `FragmentSpec` type has the following fields:

Field	Description	Required/Optional
displayName	DisplayName is a short descriptive name used for a Fragment.	Optional
git	Defines the fragment source Git repository.	Required
git.url	The repository URL, can be a HTTP/S or SSH address.	Required
git.ignore	Overrides the set of excluded patterns in the .sourceignore format (which is the same as .gitignore). If not provided, a default of `.git/` is used.	Optional (**)
git.interval	The interval at which to inquire for repository updates. If not provided the default is 10 min.	Optional (**)
git.ref	Git reference to checkout and monitor for changes, the default is main branch.	Optional (**)
git.ref.branch	The Git branch to checkout, defaults to main.	Optional (**)
git.ref.commit	The Git commit SHA to checkout, if specified tag filters are ignored.	Optional (**)

Field	Description	Required/Optional
git.ref.semver	The Git tag semver expression, takes precedence over tag.	Optional (**)
git.ref.tag	The Git tag to checkout, takes precedence over branch.	Optional (**)
git.secretRef	The secret name containing the Git credentials. For HTTPS repositories, the secret must contain user name and password fields. For SSH repositories, the secret must contain identity, identity.pub, and known_hosts fields.	Optional (**)
git.subPath	SubPath is the directory inside the Git repository to consider as the root of the accelerator or fragment. Defaults at the root of the repository.	Optional

* Any optional text boxes marked with an asterisk (*) are populated from a text box of the same name in the `accelerator` definition in the `accelerator.yaml` file if that is present in the Git repository for the accelerator.

** Any fields marked with a double asterisk (**) are part of the Flux GitRepository CRD that is documented in the Flux Source Controller Git Repositories documentation.

*** Any fields marked with a triple asterisk (`***`) are optional but either `git` or `source` is required to specify the repository to use. If `git` is specified, the `git.url` is required, and if `source` is specified, `source.image` is required.

Excluding files

The `git.ignore` field defaults to `.git/`, which is different from the defaults provided by the Flux Source Controller GitRepository implementation. You can override this, and provide your own exclusions. For more information, see fluxcd/source-controller Excluding files.

Test accelerators in Application Accelerator

This topic tells you how to test an updated accelerator, or fragment that is not registered in your Tanzu Application Platform (commonly known as TAP) cluster.

Generating a project from local sources

When you are authoring your accelerator, you can test it before committing any changes.

With the `tanzu accelerator generate-from-local` command, you can run your accelerator (or fragment), including any changes you have locally, specify a set of options and view the generated project.

You can run the accelerator using the components on your Tanzu Application Platform cluster, without impacting the state of the Tanzu Application Platform cluster.

To do so, ensure that you have the following prerequisites:

- The Tanzu CLI is installed, with the Application Accelerator plug-in. For details about installing the Tanzu CLI and plug-ins, see Tanzu CLI.
- The server URL is pointing to the Tanzu Application Platform cluster you want to test with. For details about setting the server URL, see Application Accelerator CLI plug-in overview.

For example, to use the accelerator that is located at the path `workspace/java-rest`:

```
tanzu accelerator generate-from-local --accelerator-path java-rest=workspace/java-rest
--fragment-names tap-workload,java-version --options '{"projectName":"test"}' --output
-dir generated-project
```

This generates the project in the local directory `generated-project`, using the accelerator located at `workspace/java-rest`, the fragments `tap-workload` and `java-version` which are assumed to be already registered in the Tanzu Application Platform cluster and the option `projectName` set to `test`.

For example, to use the fragment named `java-version` that is located at the path `workspace/version`:

```
tanzu accelerator generate-from-local --accelerator-name java-rest --fragment-paths ja
va-version=workspace/version --fragment-names tap-workload --options '{"projectNam
e":"test"}' --output-dir generated-project
```

This generates the project in the local directory `generated-project`, using the accelerator `java-rest` and the fragment `tap-workload` which are assumed to be already registered in the Tanzu Application Platform cluster, the fragment named `java-version` located at `workspace/version`, and the option `projectName` set to `test`.

For the full documentation for the `generate-from-local` command, see reference Tanzu accelerator generate-from-local.

No changes are made to the Tanzu Application Platform cluster that is provided with the server URL. No new accelerators/fragments are registered or modified. A Tanzu Application Platform cluster is required to ensure that there is consistency between the version that is used for testing and the version that is used when the accelerator is registered. Furthermore, it allows using registered fragments and accelerators as dependencies for the local accelerator/fragment.

CI/CD Pipeline

As you iterate on an accelerator, you can have some automated assertions run before any changes to the accelerator are accepted.

The process for generating a project from the committed source files is the same as described earlier.

When the generated project is available, you can run various assertions on it:

```
cd generated-project
test -f build.gradle
./gradlew test
```

If you have multiple assertions, you might choose to run a predefined script:

```
cd generated-project
../assertions/validate-generate-project.sh
```

You might choose to generate multiple projects from the same accelerator, providing different options for each and running different assertions on each generated project.

(Optional) Getting the Tanzu CLI in a CI/CD pipeline

If the Tanzu CLI is already available in your CI/CD pipeline you can skip this section.

VMware provides an example script that is agnostic to the CI/CD system it is running on. The script requires a variable named `TANZU_REFRESH_TOKEN` which holds a personal VMware Tanzu Network refresh token. To generate such a token see How to Authenticate. The script also uses `curl` and `jq`.

The script downloads artifacts compatible with Tanzu Application Platform version v1.4 and a Linux operating system. Update the script to suit the Tanzu Application Platform version and OS that you are using.

```
#!/bin/bash

# Get access token using personal Tanzu Network refresh token
# See https://network.tanzu.vmware.com/docs/api#how-to-authenticate
ACCESS_TOKEN=$(curl -X POST https://network.tanzu.vmware.com/api/v2/authentication/acc
ess_tokens -d '{"refresh_token":"'"$TANZU_REFRESH_TOKEN"'"}' | jq -r ".access_token")

# Download bundle
# See https://docs.vmware.com/en/VMware-Tanzu-Application-Platform/1.5/tap/GUID-instal
l-tanzu-cli.html#cli-plugin-install
# Update url to download desired version
mkdir -p $HOME/tanzu
curl -L -X GET https://network.tanzu.vmware.com/api/v2/products/tanzu-application-plat
form/releases/1205491/product_files/1352407/download -H "Authorization: Bearer $ACCESS
_TOKEN" --output bundle.tar

# Unpack bundle
export TANZU_CLI_NO_INIT=true
export VERSION=v0.25.0 # Update to desired version
tar -xvf bundle.tar -C $HOME/tanzu
cd $HOME/tanzu

# Install CLI
# Update to use desired OS
sudo install cli/core/$VERSION/tanzu-core-linux_amd64 /usr/local/bin/tanzu

# Install plugins
tanzu plugin install accelerator
```

Use the Provenance transform in Application Accelerator

This topic tells you about the Application Accelerator `Provenance` transform in Tanzu Application Platform (commonly known as TAP).

The `Provenance` transform is a special transform used to generate a file that provides details of the accelerator engine transform.

The `Provenance` transform provides traceability and visibility into the generation of an application from an accelerator. The following information is embedded into a file that is part of the generated project:

- Which accelerator was used to bootstrap the project
- Which version of the accelerator was used
- When the application was bootstrapped
- Who bootstrapped the application

For more information on the structure of the file and how to enable application bootstrapping provenance, see Provenance transform.

Use the Application Accelerator Visual Studio Code extension

This topic describes how to use the Application Accelerator Visual Studio Code extension to explore and generate projects from the defined accelerators in Tanzu Application Platform (commonly known as TAP) using VS Code.

The Application Accelerator Visual Studio Code extension lets you explore and generate projects from the defined accelerators in Tanzu Application Platform using VS Code.

Dependencies

- To use the VS Code extension, the extension must access the Tanzu Application Platform GUI URL. For information about how to retrieve the Tanzu Application Platform GUI URL, see Retrieving the URL for the Tanzu Application Platform GUI.
- (Optionally) To use Git repository provisioning during project creation in the VS Code extension, you must enable GitHub repository creation in the Application Accelerator plug-in. For more information, see Create an Application Accelerator Git repository during project creation.

Installation

Use the following steps to install the Application Accelerator Visual Studio extension:

1. Sign in to VMware Tanzu Network and download the “Tanzu App Accelerator Extension for Visual Studio Code” file from the product page for VMware Tanzu Application Platform.
2. Open VS Code.

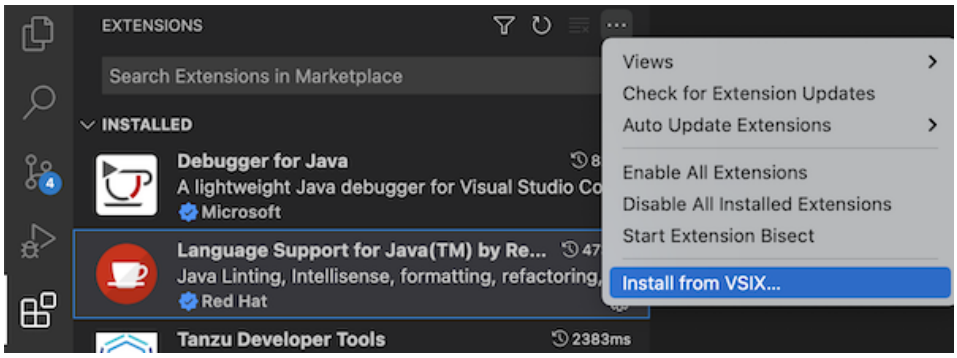
   Option 1:

   1. From the Command Palette (cmd + shift + P), run “Extensions: Install from VSIX…”.
   2. Select the extension file **tanzu-app-accelerator-<EXTENSION_VERSION>.vsix**.

   Option 2:

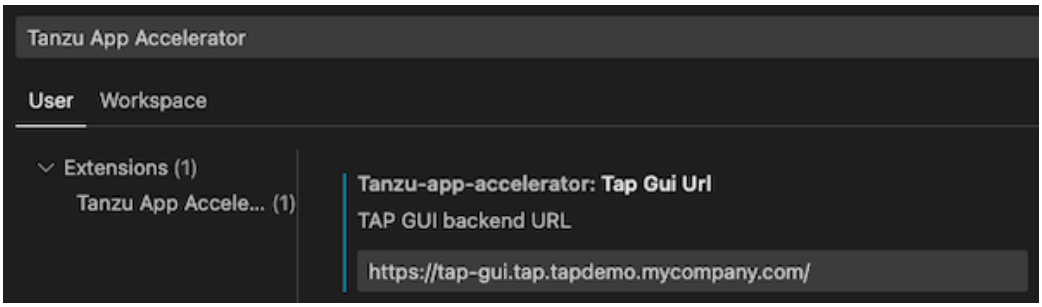
   1. Select the **Extensions** tab: 
   2. Select `Install from VSIX…` from the overflow menu.

Configure the extension

Before using the extension, you need follow the next steps:

1. Go to VS Code settings - click **Code > Preferences > Settings > Extensions > Tanzu App Accelerator**.
2. Look for the setting `Tap Gui Url`.
3. Add the `Tanzu Application Platform GUI` URL.

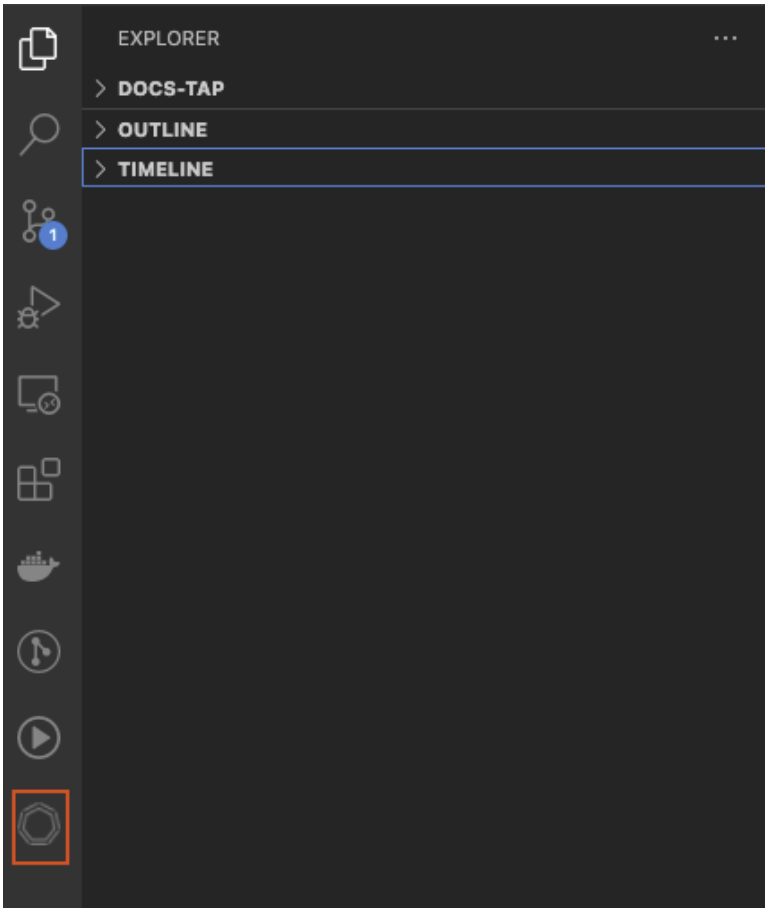


An example URL: `https://tap-gui.myclusterdomain.myorg.com`. If you have access to the Tanzu Application Platform cluster that is running the Tanzu Application Platform GUI, you can run the following command to determine the fully-qualified domain name:

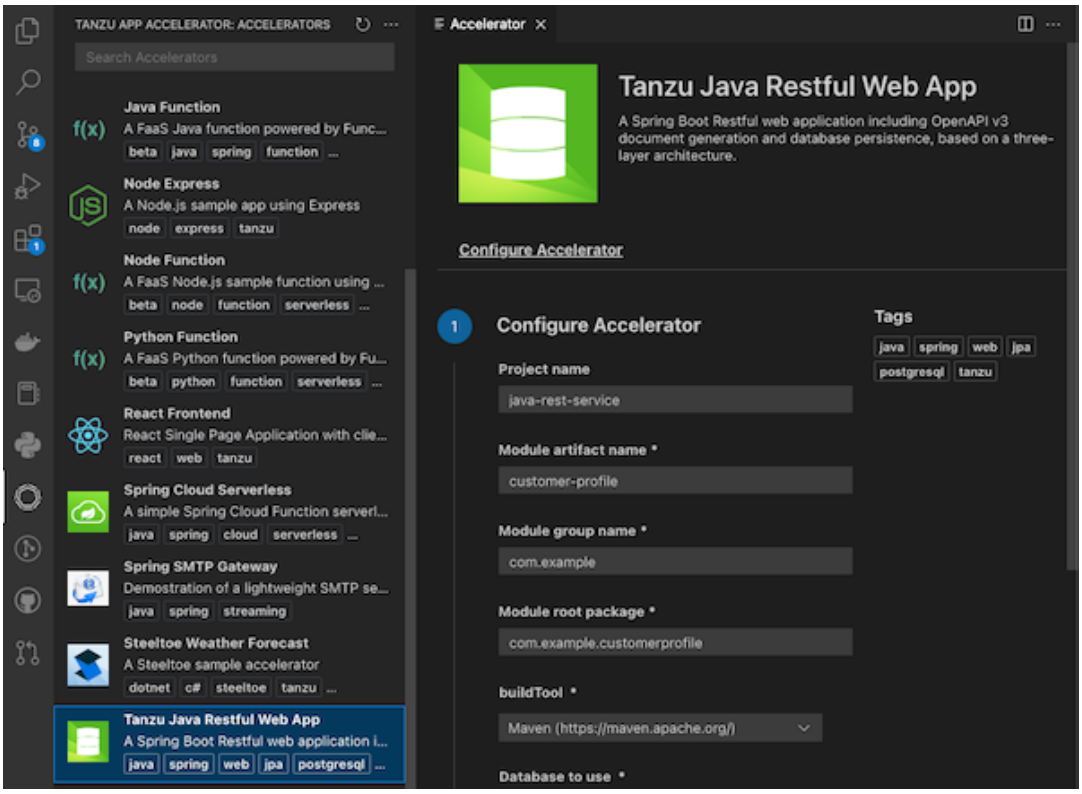
```
kubectl get httpproxy tap-gui -n tap-gui
```

Using the extension

After adding the `Tap Gui Url` you can explore the defined accelerators accessing the Application Accelerator extension icon:



Choose any of the defined accelerators, fill the options and click the `generate project`



Retrieving the URL for the Tanzu Application Platform GUI

If you have access to the Tanzu Application Platform cluster that is running the Tanzu Application Platform GUI, you can run the following command to determine the fully-qualified domain name:

```
kubectl get httpproxy tap-gui -n tap-gui
```

With an expected response of something similar to:

```
NAME      FQDN                                    TLS SECRET     STATUS   STATUS DES
CRIPTION
tap-gui   tap-gui.tap.tapdemo.myorg.com           tap-gui-cert   valid    Valid HTTP
Proxy
```

Download and Install Self-Signed Certificates from the Tanzu Application Platform GUI

To enable the Application Accelerator extension for VS Code to communicate with a Tanzu Application Platform GUI instance that is secured using TLS, you must download and install the certificates locally.

Prerequisites

yq is required to process the YAML output.

Procedure

1. Find the name of the Tanzu Application Platform GUI certificate. The name of the certificate might look different to the following example.

   ```
   kubectl get secret -n cert-manager
   ```

```
NAME                                              TYPE
DATA   AGE
canonical-registry-secret                         kubernetes.io/dockerconfigjson
1      18d
cert-manager-webhook-ca                           Opaque
3      18d
postgres-operator-ca-certificate                  kubernetes.io/tls
3      18d
tanzu-sql-with-mysql-operator-ca-certificate      kubernetes.io/tls
3      18d
tap-ingress-selfsigned-root-ca                    kubernetes.io/tls
3      18d <------- This is the certificate that is needed
```

2. Download the certificate:

```
kubectl get secret -n cert-manager tap-ingress-selfsigned-root-ca -o yaml | yq
'.data."ca.crt"' | base64 -d > ca.crt
```

3. Install the certificate on your local system and fully restart any applications that uses the certificate. After restarting, the application uses the certificate to communicate with the endpoints using TLS. For more information, see Installing a root CA certificate in the trust store in the Ubuntu documentation.

macOS
Run:

```
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.k
eychain ca.crt
```

Windows
Complete the following steps:

1. Use Windows Explorer to navigate to the directory where the certificate was downloaded and click on the certificate.
2. In the Certificate window, click **Install Certificate...**.
3. Change the **Store Location** from **Current User** to **Local Machine**. Click **Next**.
4. Select **Place all certificates in the following store**, click **Browse**, and select **Trusted Root Certification Authorities**
5. Click **Finish**.
6. A pop-up window stating **The import was successful.** is displayed.

Use the Application Accelerator IntelliJ plug-in

This topic tells you about the Application Accelerator IntelliJ plug-in. The plug-in is used to explore and generate projects from the defined accelerators in Tanzu Application Platform (commonly called TAP) using IntelliJ.

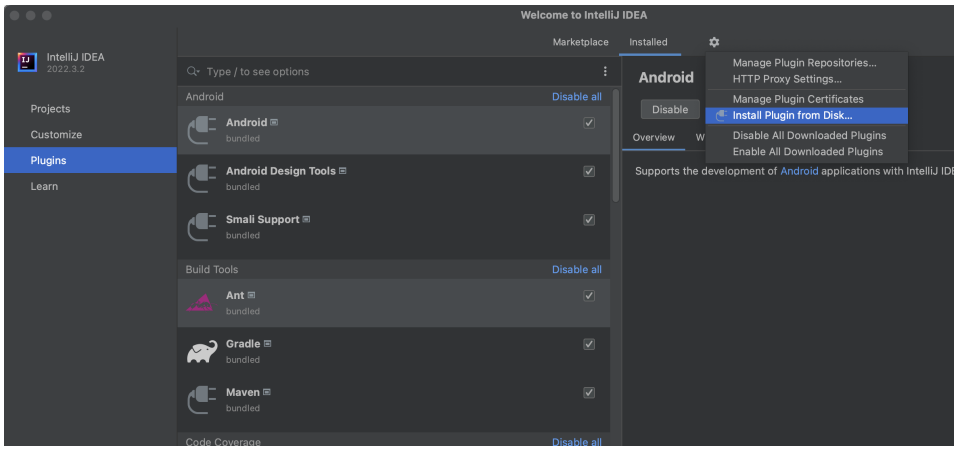
Dependencies

The plug-in must have access to the Tanzu Application Platform GUI URL. For information about how to retrieve the Tanzu Application Platform GUI URL, see Retrieving the URL for the Tanzu Application Platform GUI.

Installation

Use the following steps to install the Application Accelerator IntelliJ plug-in:

1. Sign in to VMware Tanzu Network and download the Tanzu App Accelerator Extension for Intellij file.
2. Open IntelliJ
    1. From the `Plugins` section, select the Gear button and select **Install Plugin from Disk...**



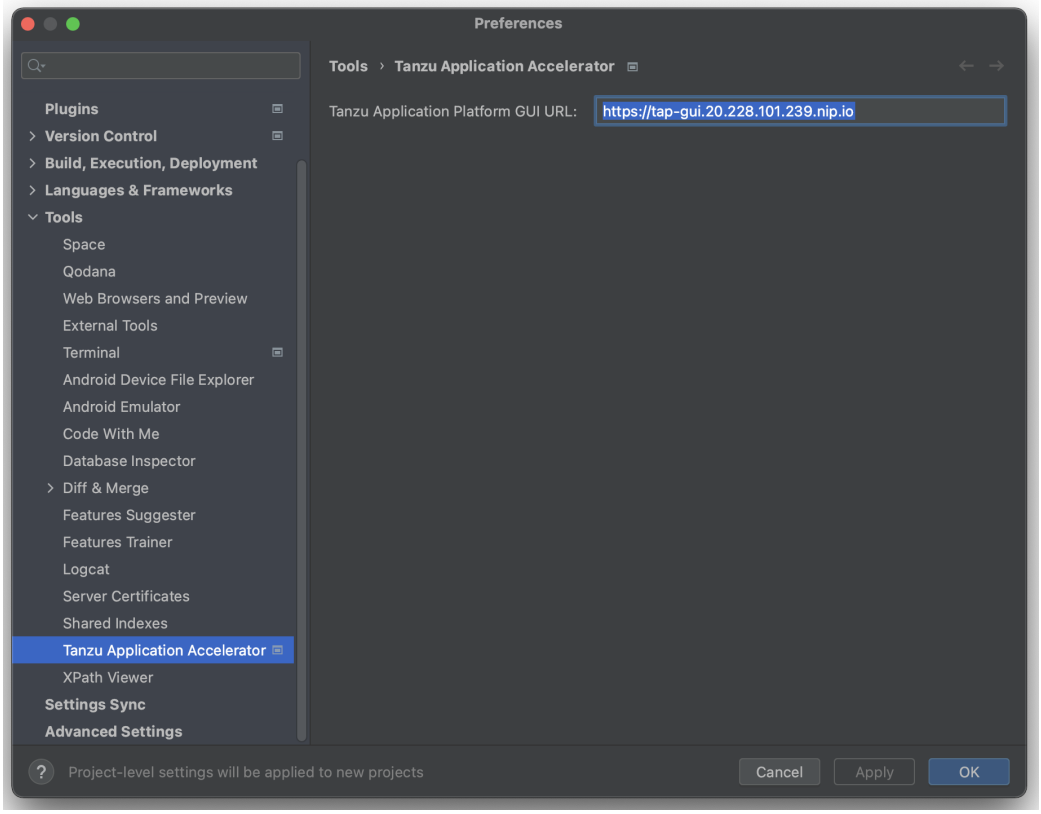
    2. Select the plug-in zip file and restart IntelliJ.

Configure the plug-in

Before using the plug-in, you must enter the Tanzu Application Platform GUI URL in the IntelliJ Preferences:

1. Go to the IntelliJ menu, select **IntelliJ IDEA > Preferences** > **Tools** > **Tanzu Application Accelerator**.
2. Add the Tanzu Application Platform GUI URL. For example, `https://tap-gui.myclusterdomain.myorg.com`. If you have access to the Tanzu Application Platform cluster that is running the Tanzu Application Platform GUI, run the following command to determine the fully-qualified domain name:

```
kubectl get httpproxy tap-gui -n tap-gui
```

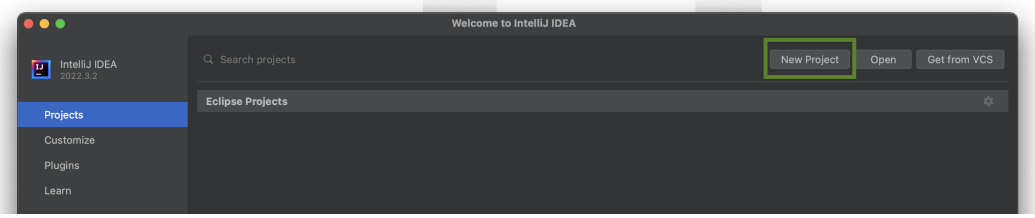


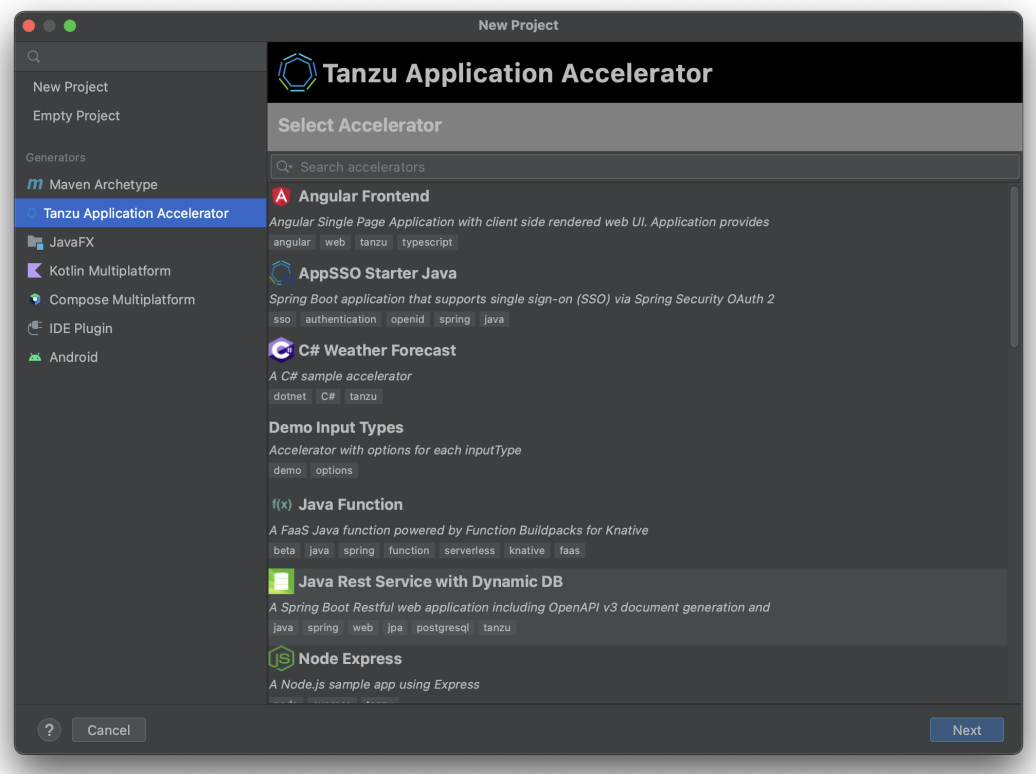
3. Click **Apply** and **OK**.

Using the plug-in

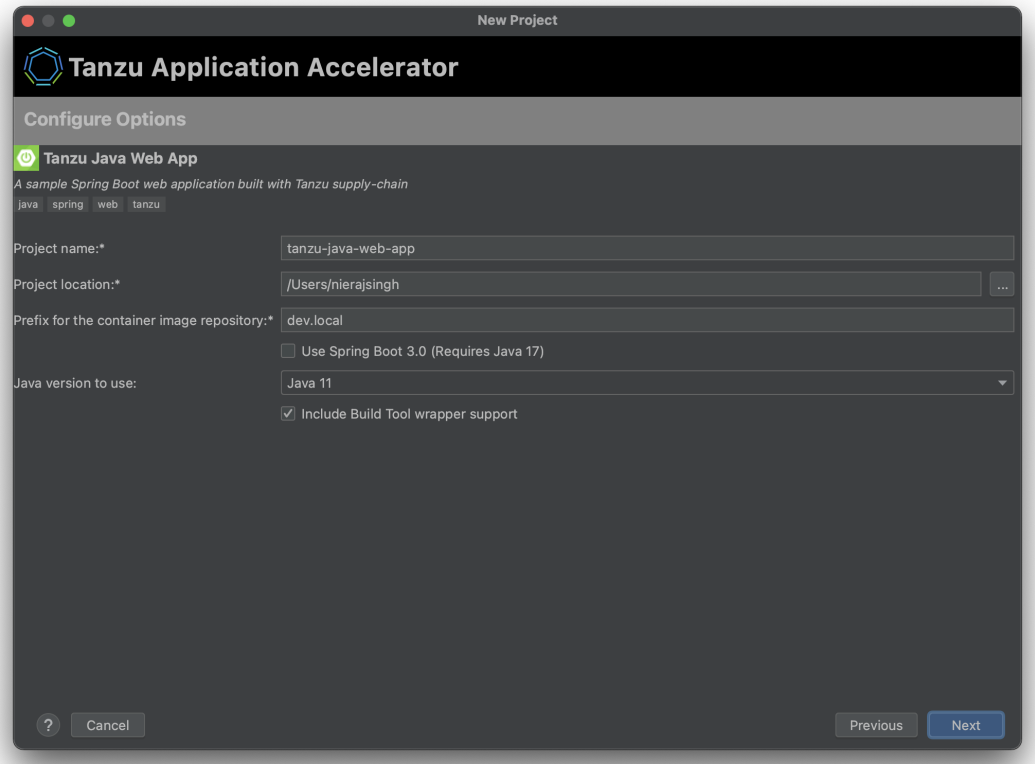
After adding the Tanzu Application Platform GUI URL, you can explore the defined accelerators:

1. Select **New Project**, then select **Tanzu Application Accelerator**.

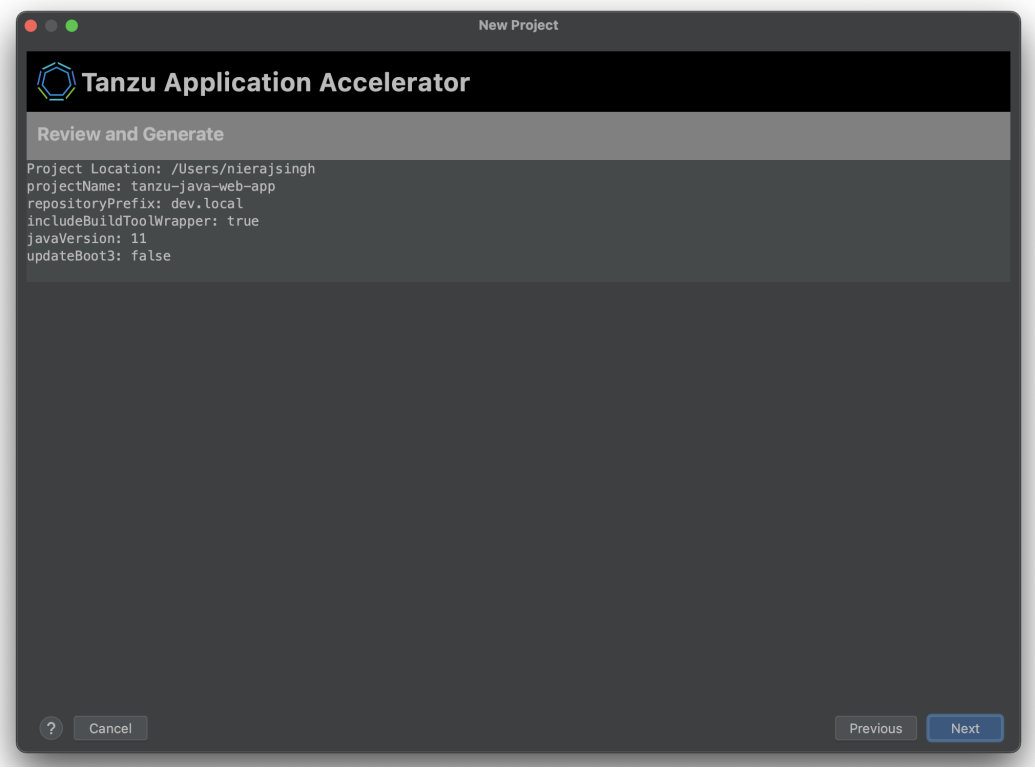




2. Choose one of the defined accelerators and configure the options.



3. Click `Next` to review the options.



4. Click `Next` to download the project. If the project is downloaded successfully, the `Create` button is enabled and you can now create and open the project.

Retrieving the URL for the Tanzu Application Platform GUI

If you have access to the Tanzu Application Platform cluster that is running the Tanzu Application Platform GUI, run the following command to determine the fully-qualified domain name:

```
kubectl get httpproxy tap-gui -n tap-gui
```

The result is similar to:

```
NAME      FQDN                                TLS SECRET     STATUS   STATUS DES
CRIPTION
tap-gui   tap-gui.tap.tapdemo.myorg.com       tap-gui-cert   valid    Valid HTTP
Proxy
```

Download and Install Self-Signed Certificates

To enable communication between the Application Accelerator plug-in and a Tanzu Application Platform GUI instance that is secured using TLS, you must download and install the certificates locally.

Prerequisites

yq is required to process the YAML output.

1. Find the name of the Tanzu Application Platform GUI certificate. The name of the certificate might look different to the following example.

   ```
   kubectl get secret -n cert-manager
   ```

```
NAME                                              TYPE
DATA   AGE
canonical-registry-secret                         kubernetes.io/dockerconfigjson
1      18d
cert-manager-webhook-ca                           Opaque
3      18d
postgres-operator-ca-certificate                  kubernetes.io/tls
3      18d
tanzu-sql-with-mysql-operator-ca-certificate      kubernetes.io/tls
3      18d
tap-ingress-selfsigned-root-ca                    kubernetes.io/tls
3      18d <------- This is the certificate that is needed
```

2. Download the certificate:

```
kubectl get secret -n cert-manager tap-ingress-selfsigned-root-ca -o yaml | yq
'.data."ca.crt"' | base64 -d > ca.crt
```

3. Install the certificate on your local system and fully restart any applications that leverage the certificate. After restarting, the application uses the certificate to communicate with the endpoints using TLS.

MacOS
Run:

```
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.k
eychain ca.crt
```

For more information, see Installing a root CA certificate in the trust store in the Ubuntu documentation.

Windows
Complete the following steps:

1. Using Windows Explorer, navigate to the directory where the certificate was downloaded and double-click on the certificate.
2. In the Certificate window, click **Install Certificate...**.
3. Change the **Store Location** from **Current User** to **Local Machine**. Click **Next**.
4. Select **Place all certificates in the following store**, click **Browse**, and select **Trusted Root Certification Authorities**
5. Click **Finish**.
6. A pop-up window stating **The import was successful.** is displayed.

Application Accelerator best practices

The following topics tells you about best practices for authoring accelerators and fragments.

- **Best practices for using Accelerators**

  A collection of best practices for authoring accelerators.

- **Best practices for using Fragments**

  A collection of best practices for authoring fragments.

Best practices for using accelerators

This topic tells you about the benefits, and design considerations for accelerators.

Benefits of using an accelerator

There are several good reasons to develop accelerators:

- If you're repeatedly using the same application architecture for new applications.
- To enforce standardization of technology stacks and application setups throughout your organization.
- To share best practices around application architecture, application, and test setup.

Design considerations

Each accelerator must have only one base technology stack, combined with related tooling, and one target architecture. For example, if you use both Spring Boot and C# .NET Core applications in your target environment, you must set up two separate accelerators. Mixing multiple technology stacks and multiple target architectures makes both the directory structure and acceleratory.YAML unreadable.

The scope of your accelerator must align with your different types of deployments. For example, back-end API, front-end UI, business service, and so on.

Choose OpenRewrite-based transformation over ReplaceText-based transformation when possible. OpenRewrite-based transformations understand the semantics of the files they work on, for example, Maven pom.xml or Java source files. OpenRewrite-based transformations also provide more accurate and robust modifications. As a last resort, ReplaceText supports a regex mode. When used with capturing groups in the replacement string, ReplaceText allows most modifications.

Housekeeping rules

VMware has found that the following rules keep the set of accelerators clear and findable for end users:

- Use an intuitive name and short description that reflects the accelerators purpose. The word 'accelerator' must not be in the name.
- Use an appropriate and intuitive icon.
- Use tags that reflect language, framework, and type of service. For example, database, messaging, and so on. This helps when searching for an accelerator by tags. Tag names must use lowercase letters, consist of [a-z0-9+#] separated by [-], and not exceed 63 characters.
- Accelerators must expose options to allow configuring an accelerator for different use cases instead of creating multiple similar accelerators.
- Options must be straightforward, the description of each clearly stating the role it plays in the accelerator. Options must have default values when appropriate.
- Options must be short so that they are easy to navigate. Make options conditional on other options as appropriate.
- Free text options that have limitations on their values must ensure these limitations are met by a regular expression-based validation. This validation ensures early feedback on invalid user input.
- Generated application skeletons must have a detailed README file that describes the function and structure of a generated application. It must provide detailed information

about how developers can build and deploy a generated application of the accelerator and how to use it.

Tests

Application skeleton

An accelerator that generates an application skeleton without a good test suite for the different layers of the application promotes bad behavior. It could result in code running in production without testing.

Tests you could use for the application skeleton:

- An overall application test that bootstraps the application to see if it comes online.
- A test per layer of the application. For example, presentation layer, business layer, and data layer. These tests can be unit tests that leverage stubbing or mocking frameworks.
- An integration test per layer of the application, especially the presentation and data layer. For example, you can provide an integration test with some database interaction by using test containers.

Best practices for using fragments

This topic tells you about the benefits, and design considerations for fragments.

Benefits of using Fragment

A fragment is a partial accelerator. It can do the same transformations as an accelerator, but it cannot run on its own. It's always part of the calling (host) accelerator.

Developing a fragment is useful in the following situations:

- When you must update a version of an element of a technology stack in multiple locations. For example, when the Java Development Kit (JDK) version must be updated in the build tool configuration, the buildpack configuration, and in the deployment options.
- To add a consistent cross-cutting concern to a set of accelerators. For example, logging, monitoring, or support for a certain type of deployment or framework.
- To add integration with some technology to a generated application skeleton. For example, certain database support, support for a messaging middleware, or integration with an email provider.

Design considerations

Developing and maintaining a fragment is complex. The following is a list of design considerations:

- The fragment you develop must work with all possible syntax and format variations. For example, dependency in a `Gradle` build.gradle.kts can have the following forms:
    - `implementation('org.springframework.boot:spring-boot-starter')`
    - `implementation("org.springframework.boot:spring-boot-starter")`
    - `implementation(group = "org.springframework.boot", name= "spring-boot-starter")`
    - `implementation(group = 'org.springframework.boot', name= 'spring-boot-starter')`

    - `implementation(name= “spring-boot-starter", group = "org.springframework.boot”)`
- The fragment can be used in multiple accelerator contexts and its behavior must result in a compilable and deployable application skeleton.
- Testing a fragment in isolation is more difficult than testing an accelerator. Testing takes more time because all the combinations must be tested from an accelerator perspective.
- When flexibly reusing fragments in different combinations, each fragment must cover a small, cohesive function. Fragments must follow these two UNIX principles:
    - Small is beautiful.
    - Each fragment does one thing well.
- Keep the files the fragment changes to a minimum. Only change the files that are related to the same technology stack for the same purpose.
- The design of both the accelerator and fragment is limited by the technology stack and the target deployment technology chosen for the accelerator. For example, to create a fragment for standardizing logging, you must create one fragment per base technology stack.

Housekeeping rules

Fragments are used by accelerator authors. VMware has found that the following guidelines keep fragments understandable and reusable.

- Give fragments an intuitive name and short description that reflects their purpose. Do not include “fragment” in the name.
- Fragments must expose options to allow configuring the output of execution.
- Each fragment must contain a README file explaining the additional functions the fragment adds to a generated application skeleton. List any options expected by this fragment. Also describe how this fragment can be included in a host accelerator. Be sure to state any known limitations or use cases not covered. For example, if the fragment supports Maven and Gradle as build tools but only Groovy DSL of Gradle is supported, the README file must include this information.
- If a fragment must provide additional documentation to end users, it can either be added to a README-X file of the generated application skeleton or append a section to the host’s README.

Troubleshoot Application Accelerator

This topic provides troubleshooting steps for development, accelerator authorship, and operations issues in Application Accelerator.

Installation issues

Depending on the error output, you can take the following actions to troubleshoot installation problems.

Verify installed packages

The package might be already installed. Verify this by running:

```
tanzu package installed list -n tap-install
```

Look for any package called `accelerator.apps.tanzu.vmware.com`.

Look at resource events

The error might be within the custom resources such as accelerator, Git repository, fragment, and so on. Find these errors by using the Kubernetes command line interface tool (kubectl).

Here is an example using the custom resource `accelerator`:

1. Check for errors. For example, review the custom resource `accelerator` by running:

   ```
   kubectl get acc -n accelerator-system
   ```

   Note items in the output with a `READY` status `False`:

   ```
   NAME                        READY   REASON      AGE
   appsso-starter-java         True    Ready       5h2m
   where-for-dinner            True    Ready       5h2m
   java-function               True    Ready       5h2m
   java-rest-service           True    Ready       5h2m
   java-server-side-ui         True    Ready       5h2m
   node-express                True    Ready       5h2m
   node-function               False   Not-Ready   5h2m
   python-function             True    Ready       5h2m
   spring-cloud-serverless     True    Ready       5h2m
   spring-smtp-gateway         True    Ready       5h2m
   tanzu-java-web-app          True    Ready       5h2m
   tap-initialize              True    Ready       5h2m
   weatherforecast-csharp      True    Ready       5h2m
   weatherforecast-steeltoe    True    Ready       5h2m
   ```

2. To see more information about any error events you found, run:

   ```
   kubectl get acc node-function -n accelerator-system -o yaml
   ```

   Look at the event section for more information about the error.

Development issues

Failure to generate a new project

`URI is not absolute` error

The `generate` command fails with the following error:

```
% tanzu accelerator generate test --server-url https://accelerator.example.com
Error: there was an error generating the accelerator, the server response was: "URI is
not absolute"

Use:
  tanzu accelerator generate [flags]

Examples:
  tanzu accelerator generate <accelerator-name> --options '{"projectName":"test"}'

Flags:
  -h, --help                  help for generate
      --options string        options JSON string
      --options-file string   path to file containing options JSON string
      --output-dir string     directory that the zip file will be written to
      --server-url string     the URL for the Application Accelerator server
```

```
Global Flags:
      --context name      name of the kubeconfig context to use (default is current-co
ntext defined by kubeconfig)
      --kubeconfig file   kubeconfig file (default is $HOME/.kube/config)

there was an error generating the accelerator, the server response was: "URI is not ab
solute"

Error: exit status 1

✖  exit status 1
```

This indicates that the accelerator resource requested is not in a `READY` state. Review the instructions in the When Accelerator ready column is false section or contact your system admin.

Accelerator authorship issues

General tips

Speed up the reconciliation of the accelerator

Set the `git.interval` to make the accelerator reconcile sooner. The default interval is 10 minutes, which is too long when developing an accelerator.

You can set this when using the YAML manifest:

```
apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  name: test-accelerator
spec:
  git:
    url: https://github.com/trisberg/test-accelerator
    ref:
      branch: main
    interval: 10s
```

You can also set this when creating the accelerator resource. To do so from the Tanzu CLI, run:

```
tanzu accelerator create test-accelerator --git-repo https://github.com/trisberg/test-
accelerator --git-branch main --interval 10s
```

Use a source image with local accelerator source directory

You don't have to use a Git repository when developing an accelerator. You can create an accelerator based on content in a local directory using `--local-path` when creating the accelerator resource.

Push the local path content to an OCI image by running:

```
tanzu accelerator create test-accelerator --local-path . --source-image REPO-PREFIX/te
st-accelerator --interval 10s
```

Where `REPO-PREFIX` is your own repository prefix. Use a repository that the deployed Application Accelerator system can access.

The interval is 10s so that you can push changes to the source-image repository and get faster reconcile time for the accelerator resource. When you have made changes to your accelerator source, push those changes by running:

```
tanzu accelerator push --local-path . --source-image REPO-PREFIX/test-accelerator
```

Where `REPO-PREFIX` is your own repository prefix. Use a repository that is accessible to the deployed Application Accelerator system.

Expression evaluation errors

Expression evaluation errors include:

- Expression `evaluated to null`, such as:

  ```
  Could not read response from accelerator: java.lang.IllegalArgumentException: E
  xpression '#mytestexp' evaluated to null
  ```

  In most cases, a typo in the variable name causes this error. Compare the expression with the defined options or any variables declared with `let`.

- `could not parse SpEL expression`, such as:

  ```
  Could not read response from accelerator: Error reading manifest:could not pars
  e SpEL expression at [Source: (InputStreamReader); line: 65, column: 1] (throug
  h reference chain: com.vmware.tanzu.accelerator.engine.manifest.Manifest["engin
  e"]->com.vmware.tanzu.accelerator.engine.transform.transforms.Combo["let"]->jav
  a.util.ArrayList[0]->com.vmware.tanzu.accelerator.engine.transform.transforms.L
  et$DerivedSymbol["expression"])
  ```

  In most cases, an error in a `let` expression causes this error. Review the error message and, for more information, see SpEL samples.

- `SpelEvaluationException`, such as:

  ```
  Could not read response from accelerator: org.springframework.expression.spel.S
  pelEvaluationException: EL1007E: Property or field 'test' cannot be found on nu
  ll
  ```

  In most cases, an error in a transform expression causes this error. Review the error message and, for more information, see SpEL samples.

Operations issues

Accelerator persists in Tanzu Application Platform GUI after deletion

If an accelerator still displays in the Tanzu Application Platform GUI after it is deleted using the `tanzu accelerator delete` command, complete the following steps to delete:

1. Navigate to your instance of the Tanzu Application Platform GUI.
2. Search for the accelerator which should be deleted and select it.
3. On the top right of the window, click the three dots, and select **Unregister Template**.

The accelerator is not longer displayed in the Tanzu Application Platform GUI Accelerator Catalog.

Check status of accelerator resources

Verify the status of accelerator resources by using kubectl or the Tanzu CLI:

- From kubectl, run:

  ```
  kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-syste
  m
  ```

- From the Tanzu CLI, run:

```
tanzu accelerator list
```

Verify that the `READY` status is `true` for all accelerators.

When Accelerator ready column is blank

1. View the status of `accelerator-system` by running:

   ```
   kubectl get deployment -n accelerator-system
   ```

   Example output:

   ```
   NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
   acc-engine                        1/1     1            1           3d5h
   acc-server                        1/1     1            1           2d1h
   accelerator-controller-manager    0/1     1            0           3d5h
   ```

2. View the logs for any component with no Pods available by running:

   ```
   kubectl logs deployment/COMPONENT-NAME/ -n accelerator-system -p
   ```

   Where `COMPONENT-NAME` is the component with no pods you retrieved in the previous step.

   - If the log has the following error then the Flux CD source-controller is not installed:

     ```
     2021-11-18T20:55:18.963Z ERROR setup problem running manager {"error": "f
     ailed to wait for accelerator caches to sync: no matches for kind \"GitRe
     pository\" in version \"source.toolkit.fluxcd.io/v1beta1\""}
     ```

   - If the log has the following error, the Tanzu Application Platform source-controller is not installed:

     ```
     2021-11-18T20:50:10.557Z ERROR setup problem running manager {"error": "f
     ailed to wait for accelerator caches to sync: no matches for kind \"Image
     Repository\" in version \"source.apps.tanzu.vmware.com/v1alpha1\""}
     ```

When Accelerator ready column is false

View the `REASON` column for non-ready accelerators. Run:

```
kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
```

REASON: `GitRepositoryResolutionFailed`

For example:

```
$ kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
NAME        READY   REASON                          AGE
more-fun    False   GitRepositoryResolutionFailed   28s
```

1. View the resource status. Run:

   ```
   kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelerato
   r-system hello-fun
   ```

2. Read `status.conditions.message` near the end of the output to learn the likely cause of failure. For example:

```
status:
  address:
    url: http://accelerator-engine.accelerator-system.svc.cluster.local/invocat
ions
  artifact:
    message: 'unable to clone ''https://github.com/vmware-tanzu/application-acc
elerator-samples'',
      error: couldn''t find remote ref "refs/heads/test"'
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T21:05:47Z"
    message: |-
      failed to resolve GitRepository
      unable to clone 'https://github.com/vmware-tanzu/application-accelerator-
samples', error: couldn't find remote ref "refs/heads/test"
    reason: GitRepositoryResolutionFailed
    status: "False"
    type: Ready
  description: Test-git
  observedGeneration: 1
```

In this example, `couldn't find remote ref "refs/heads/test"` reveals that the branch or tag specified doesn't exist.

Another common problem is that the Git repository doesn't exist. For example:

```
status:
  address:
    url: http://accelerator-engine.accelerator-system.svc.cluster.local/invocat
ions
  artifact:
    message: 'unable to clone ''https://github.com/vmware-tanzu/application-acc
elerator-sampl'',
      error: authentication required'
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T21:09:52Z"
    message: |-
      failed to resolve GitRepository
      unable to clone 'https://github.com/vmware-tanzu/application-accelerator-
sampl', error: authentication required
    reason: GitRepositoryResolutionFailed
    status: "False"
    type: Ready
  description: Test-git
  observedGeneration: 1
```

An error message about failed authentication might display because the Git repository doesn't exist. For example:

```
unable to clone 'https://github.com/vmware-tanzu/application-accelerator-samp
l', error: authentication required
```

REASON: `GitRepositoryResolutionPending`

For example:

```
$ kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
NAME        READY   REASON                           AGE
more-fun    False   GitRepositoryResolutionPending   28s
```

1. See the resource status. Run:

```
kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelerato
r-system hello-fun
```

2. Locate `status.conditions` at the end of the output. For example:

```
status:
  address:
    url: http://accelerator-engine.accelerator-system.svc.cluster.local/invocat
ions
  artifact:
    message: ""
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T20:17:38Z"
    message: GitRepository not yet resolved
    reason: GitRepositoryResolutionPending
    status: "False"
    type: Ready
  description: Test-git
  observedGeneration: 1
```

3. Verify that the Flux system is running and that the `READY` column has `1/1`. Run:

```
kubectl get -n flux-system deployment/source-controller
```

Example output:

```
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
source-controller   1/1     0            0           5d4h
```

REASON: `ImageRepositoryResolutionPending`

For example:

```
$ kubectl get accelerators.accelerator.apps.tanzu.vmware.com -n accelerator-system
NAME        READY   REASON                            AGE
more-fun    False   ImageRepositoryResolutionPending   28s
```

1. See the resource status. Run:

```
kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelerato
r-system hello-fun
```

2. Locate `status.conditions` at the end of the output. For example:

```
$ kubectl get -oyaml accelerators.accelerator.apps.tanzu.vmware.com -n accelera
tor-system more-fun

apiVersion: accelerator.apps.tanzu.vmware.com/v1alpha1
kind: Accelerator
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"accelerator.apps.tanzu.vmware.com/v1alpha1","kind":"Accele
rator","metadata":{"annotations":{},"name":"more-fun","namespace":"accelerator-
system"},"spec":{"description":"Test-image","source":{"image":"trisberg/more-fu
n-source"}}}
  creationTimestamp: "2021-11-18T20:32:36Z"
```

```
  generation: 1
  name: more-fun
  namespace: accelerator-system
  resourceVersion: "605401"
  uid: 407b565d-14aa-44fe-ad8d-c9b3c3a7e5ce
spec:
  description: Test-image
  source:
    image: trisberg/more-fun-source
status:
  address:
    url: http://accelerator-engine.accelerator-system.svc.cluster.local/invocat
ions
  artifact:
    message: ""
    ready: false
    url: ""
  conditions:
  - lastTransitionTime: "2021-11-18T20:32:36Z"
    message: ImageRepository not yet resolved
    reason: ImageRepositoryResolutionPending
    status: "False"
    type: Ready
  description: Test-image
  observedGeneration: 1
```

3. Verify that Tanzu Application Platform source-controller system is running and the `READY` column has `1/1`. Run:

   ```
   kubectl get -n source-system deployment/source-controller-manager
   ```

   Expected output:

   ```
   NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
   source-controller-manager   1/1     0            0           5d5h
   ```

Overview of Application Configuration Service for VMware Tanzu

Application Configuration Service (commonly known as App Config Service) provides a Kubernetes-native experience to enable the runtime configuration of existing Spring applications that were previously leveraged by using Spring Cloud Config Server.

Spring Cloud Config Server was an essential component in microservices architectures for providing runtime configuration to Spring Boot applications.

Spring Cloud Config Server did this by enabling configuration management to be hosted in Git repositories on different branches and in directories that could be used to generate runtime configuration properties for applications.

Application Configuration Service is compatible with the existing Git repository configuration management approach. It filters runtime configuration for any application by using slices that produce secrets.

For more information about Application Configuration Service, see the Application Configuration Service for VMware Tanzu documentation.

Overview of Application Configuration Service for VMware Tanzu

Application Configuration Service (commonly known as App Config Service) provides a Kubernetes-native experience to enable the runtime configuration of existing Spring applications that were previously leveraged by using Spring Cloud Config Server.

Spring Cloud Config Server was an essential component in microservices architectures for providing runtime configuration to Spring Boot applications.

Spring Cloud Config Server did this by enabling configuration management to be hosted in Git repositories on different branches and in directories that could be used to generate runtime configuration properties for applications.

Application Configuration Service is compatible with the existing Git repository configuration management approach. It filters runtime configuration for any application by using slices that produce secrets.

For more information about Application Configuration Service, see the Application Configuration Service for VMware Tanzu documentation.

Install Application Configuration Service for VMware Tanzu

This topic tells you how to install Application Configuration Service for VMware Tanzu (commonly known as App Config Service) from the Tanzu Application Platform package repository.

Prerequisites

Before installing Application Configuration Service, complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install Application Configuration Service on a compliant Kubernetes cluster:

1. List version information for the package by running:

   ```
   tanzu package available list application-configuration-service.tanzu.vmware.com
   --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list application-configuration-service.tanzu.vmware.c
   om --namespace tap-install
   - Retrieving package versions for application-configuration-service.tanzu.vmwar
   e.com...
   NAME                                                VERSION  RELEASED-AT
   application-configuration-service.tanzu.vmware.com  2.0.0    2023-03-08 19:00:0
   0 -0500 EST
   ```

2. (Optional) Make changes to the default installation settings by running:

   ```
   tanzu package available get application-configuration-service.tanzu.vmware.com/
   VERSION-NUMBER \
   --values-schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the number you discovered previously. For example, `2.0.0`.

3. Install the package by running:

   ```
   tanzu package install application-configuration-service \
   --package application-configuration-service.tanzu.vmware.com \
   ```

```
--version VERSION -n tap-install \
--values-file values.yaml
```

Where `VERSION` is the version you want, such as `2.0.0`. Using a `values.yaml` file is optional.

For example:

```
$ tanzu package install application-configuration-service \
--package application-configuration-service.tanzu.vmware.com \
--version 2.0.0 -n tap-install

Installing package 'application-configuration-service.tanzu.vmware.com'
Getting package metadata for 'application-configuration-service.tanzu.vmware.co
m'
Creating service account 'application-configuration-service-tap-install-sa'
Creating cluster admin role 'application-configuration-service-tap-install-clus
ter-role'
Creating cluster role binding 'application-configuration-service-tap-install-cl
uster-rolebinding'
Creating package resource
Waiting for 'PackageInstall' reconciliation for 'application-configuration-serv
ice'
'PackageInstall' resource install status: Reconciling
'PackageInstall' resource install status: ReconcileSucceeded

Added installed package 'application-configuration-service'
```

4. Verify that you installed the package by running:

```
tanzu package installed get application-configuration-service -n tap-install
```

For example:

```
$ tanzu package installed get application-configuration-service -n tap-install

NAME:                    application-configuration-service
PACKAGE-NAME:            application-configuration-service.tanzu.vmware.com
PACKAGE-VERSION:         2.0.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Overview of Application Live View

Application Live View is a lightweight insights and troubleshooting tool for app developers and app operators that helps you to look inside running applications. It is based on the concept of Spring Boot Actuators.

The application provides information from inside the running processes using endpoints, in this case, HTTP endpoints. Application Live View uses those endpoints to get and interact with the data from apps.

Value proposition

Application Live View is a diagnostic tool for developers to manage and analyze runtime characteristics of containerized apps. In addition, it provides a Kubernetes-native feel for developers to manage their apps in a Kubernetes environment more effectively.

Intended audience

This documentation is intended for developers and operators to visualize the actuator information of their running apps on Application Live View for VMware Tanzu. This documentation helps developers to monitor and troubleshoot apps in development, staging, and production environments. It is also intended to help app operators to deploy and administer containerized apps in a Kubernetes environment.

Supported application platforms

You can extend Application Live View to support multiple app platforms, including, but not limited to, Spring Boot, Spring Cloud Gateway, and Steeltoe.

Multicloud compatibility

Using Tanzu platform, you can integrate Application Live View to monitor apps running across on-premises, public clouds, and edge. The platform provides a centralized view to manage apps across cloud environments, which accelerates developer productivity and reduces time-to-market.

Deployment

Use a connector as the mode of deployment for registering apps with the Application Live View running on a Kubernetes cluster. A connector is a component responsible for discovering multiple apps running on a Kubernetes cluster and is installed as a DaemonSet by default.

Overview of Application Live View

Application Live View is a lightweight insights and troubleshooting tool for app developers and app operators that helps you to look inside running applications. It is based on the concept of Spring Boot Actuators.

The application provides information from inside the running processes using endpoints, in this case, HTTP endpoints. Application Live View uses those endpoints to get and interact with the data from apps.

Value proposition

Application Live View is a diagnostic tool for developers to manage and analyze runtime characteristics of containerized apps. In addition, it provides a Kubernetes-native feel for developers to manage their apps in a Kubernetes environment more effectively.

Intended audience

This documentation is intended for developers and operators to visualize the actuator information of their running apps on Application Live View for VMware Tanzu. This documentation helps developers to monitor and troubleshoot apps in development, staging, and production environments. It is also intended to help app operators to deploy and administer containerized apps in a Kubernetes environment.

Supported application platforms

You can extend Application Live View to support multiple app platforms, including, but not limited to, Spring Boot, Spring Cloud Gateway, and Steeltoe.

Multicloud compatibility

Using Tanzu platform, you can integrate Application Live View to monitor apps running across on-premises, public clouds, and edge. The platform provides a centralized view to manage apps across cloud environments, which accelerates developer productivity and reduces time-to-market.

Deployment

Use a connector as the mode of deployment for registering apps with the Application Live View running on a Kubernetes cluster. A connector is a component responsible for discovering multiple apps running on a Kubernetes cluster and is installed as a DaemonSet by default.

Install Application Live View

This topic tells you how to install Application Live View from the Tanzu Application Platform (commonly known as TAP) package repository.

Overview

Application Live View includes four packages you must install. The following table lists these packages and shows the Tanzu Application Platform profiles each package is included in.

Package	Profiles	Details
Application Live View APIServer (`apiserver.appliveview.tanzu.vmware.com`)	Full, Iterate, Run	Installed in the `appliveview-tokens-system` namespace
Application Live View back end (`backend.appliveview.tanzu.vmware.com`)	Full, View	Installed with Tanzu Application Platform GUI in the `app-live-view` namespace
Application Live View connector (`connector.appliveview.tanzu.vmware.com`)	Full, Iterate, Run	Installed as a DaemonSet in the `app-live-view-connector` namespace
Application Live View conventions (`conventions.appliveview.tanzu.vmware.com`)	Full, Iterate, Build	Installed in the `app-live-view-conventions` namespace

For more information about these packages, see Application Live View internal architecture.

Note

Follow the steps in this topic if you do not want to use a profile to install Application Live View. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before installing Application Live View:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Cartographer Conventions, which is bundled with Supply Chain Choreographer as of v0.5.3. To install, see Installing Supply Chain Choreographer. For more information, see Cartographer Conventions.

Install Application Live View

You can install Application Live View in single cluster or multicluster environment:

- `Single cluster`: All Application Live View components are deployed in a single cluster. The user can access Application Live View plug-in information of the applications across all the namespaces in the Kubernetes cluster. This is the default mode of Application Live View.
- `Multicluster`: In a multicluster environment, the Application Live View back end component is installed only once in a single cluster and exposes a RSocket registration for the other clusters using Tanzu shared ingress. Each cluster continues to install the connector as a DaemonSet. The connectors are configured to connect to the central instance of the Application Live View back end.

The improved security and access control secures the communication between the Application Live View components. For more information, see Configure security and access control in Application Live View.

Install Application Live View back end

To install Application Live View back end:

1. List version information for the package by running:

   ```
   tanzu package available list backend.appliveview.tanzu.vmware.com --namespace t
   ap-install
   ```

   For example:

   ```
   $ tanzu package available list backend.appliveview.tanzu.vmware.com --namespace
   tap-install

   - Retrieving package versions for backend.appliveview.tanzu.vmware.com...
     NAME                                 VERSION        RELEASED-AT
     backend.appliveview.tanzu.vmware.com  1.5.1          2023-03-29T00:00:00Z
   ```

2. Create the file `app-live-view-backend-values.yaml` using the following information:

   - **For a single-cluster environment:** The Application Live View back end is exposed through the Kubernetes cluster service.

     By default, ingress is deactivated for Application Live View back end. You do not have to change your `app-live-view-backend-values.yaml` file in this step.

   - **For a multicluster environment:** Set the flag `ingressEnabled` to `true` for the Application Live View back end to be exposed on the ingress domain.

     ```
     appliveview:
       ingressEnabled: true
     ```

   - **For a profile install using shared ingress domain key:** If you are using a Tanzu Application Platform profile installation and the top-level key `shared.ingress_domain` is set in the `tap-values.yaml`, the back end is automatically exposed through the shared ingress.

     To override the shared ingress for Application Live View in a multicluster environment, use the following values:

     ```
     appliveview:
       ingressEnabled: true
     ```

```
  ingressDomain: ${INGRESS-DOMAIN}
```

Where `INGRESS-DOMAIN` is the top-level domain you use for the `tanzu-shared-ingress` service's external IP address. The `appliveview` subdomain is prepended to the value provided.

3. Configure TLS in your `app-live-view-backend-values.yaml` file:
    - To enable TLS for Application Live View back end using a self-signed certificate:
        1. Create the `app-live-view` namespace and the TLS secret for the domain. You must do this before installing the Tanzu Application Platform packages in the cluster so that the HTTPProxy is updated with the TLS secret. To create a TLS secret, run:

            ```
            kubectl create -n app-live-view secret tls alv-cert --cert=CERT-FI
            LE --key=KEY-FILE
            ```

            Where:

            - `SECRET-NAME` is the name you want for the TLS secret for the domain, for example, `alv-cert`.
            - `CERT-FILE` is a .crt file that contains the PEM encoded server certificate.
            - `KEY-FILE` is a .key file that contains the PEM encoded server private key.

        2. Provide the following properties in your `app-live-view-backend-values.yaml`:

            ```
            appliveview:
              ingressEnabled: true
              tls:
                namespace: "NAMESPACE"
                secretName: "SECRET-NAME"
            ```

            Where:

            - `NAMESPACE` is the targeted namespace of TLS secret for the domain.
            - `SECRET-NAME` is the name of TLS secret for the domain.

            You can edit the values to suit your project needs or leave the default values as is.

            When `ingressEnabled` is `true`, the HTTPProxy object is created in the cluster.

        3. Verify the HTTPProxy object with the TLS secret by running:

            ```
            kubectl get httpproxy -A
            ```

            Expected output:

            ```
            NAMESPACE       NAME          FQDN                                T
            LS SECRET               STATUS   STATUS DESCRIPTION
            app-live-view   appliveview   appliveview.192.168.42.55.nip.io   a
            pp-live-view/alv-cert   valid    Valid HTTPProxy
            ```

    - To enable TLS for Application Live View back end using ClusterIssuer:

1. Set the `ingressEnabled` key to `true` for TLS to be enabled on Application Live View back end using ClusterIssuer. This key is set to `false` by default.

   ```
   appliveview:
     ingressEnabled: true
   ```

   TLS is then automatically enabled on Application Live View back end using the shared ClusterIssuer. The `appliveview-cert` certificate is generated by default and its issuerRef points to the `.ingress_issuer` value. The `ingress_issuer` key consumes the value `shared.ingress_issuer` from `tap-values.yaml` by default if you don't specify the `ingress_issuer` in `tap-values.yaml`.

   When `ingressEnabled` is `true`, HTTPProxy object is created in the cluster and also `appliveview-cert` certificate is generated by default in the `app_live_view` namespace. Here, the secretName `appliveview-cert` stores this certificate.

2. To verify the HTTPProxy object with the secret, run:

   ```
   kubectl get httpproxy -A
   ```

   Expected output:

   ```
   NAMESPACE        NAME           FQDN                                T
   LS SECRET          STATUS    STATUS DESCRIPTION
   app-live-view    appliveview    appliveview.192.168.42.55.nip.io    a
   ppliveview-cert    valid     Valid HTTPProxy
   ```

3. To verify the Application Live View pages in a multicluster setup, set the appropriate connector configuration in your run cluster as listed in Install Application Live View connector later in this topic.

4. (Optional) View additional changes you can make in your `app-live-view-backend-values.yaml` file by running:

   ```
   tanzu package available get backend.appliveview.tanzu.vmware.com/VERSION-NUMBER
   \
     --values-schema \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

   For example:

   ```
   $ tanzu package available get backend.appliveview.tanzu.vmware.com/1.5.1 \
       --values-schema \
       --namespace tap-install

     KEY                      DEFAULT          TYPE        DESCRIPTION
     tls.namespace            <nil>            string      The targeted namespace
   for secret consumption by the HTTPProxy.

     tls.secretName           <nil>            string      The name of secret for
   consumption by the HTTPProxy.

     ingressDomain            tap.example.com  string      Domain to be used by th
   e HTTPProxy ingress object. The "appliveview"
                                                           subdomain will be prepe
   nded to the value provided. For example:
                                                           "example.com" would bec
   ```

```
ome "appliveview.example.com".
  ingressEnabled           false            boolean     Flag for whether or not
to create an HTTPProxy for ingress.

  ingress_issuer                            string      Cluster issuer to be us
ed in App Live View Backend.
  kubernetes_distribution                   string      Kubernetes distribution
that this package is being installed on. Accepted
                                                        values: ['''',''openshi
ft'']
  kubernetes_version                        string      Optional: The Kubernete
s Version. Valid values are '1.24.*', or ''

  server.tls.crt                            string      TLS cert file
  server.tls.enabled       false            boolean     Flag to enable tls on b
ackend
  server.tls.key                            string      TLS key file
```

5. Install the Application Live View back end package by running:

```
tanzu package install appliveview \
  --package backend.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-live-view-backend-values.yaml
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

For example:

```
$ tanzu package install appliveview \
    --package backend.appliveview.tanzu.vmware.com \
    --version 1.5.1 \
    --namespace tap-install \
    --values-file app-live-view-backend-values.yaml

- Installing package 'backend.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'backend.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-tap-install-sa'
| Creating cluster admin role 'appliveview-tap-install-cluster-role'
| Creating cluster role binding 'appliveview-tap-install-cluster-rolebinding'
| Creating package resource
| Package install status: Reconciling

Added installed package 'appliveview' in namespace 'tap-install'
```

The Application Live View back end component is deployed in `app-live-view` namespace by default.

6. Verify the Application Live View back end package installation by running:

```
tanzu package installed get appliveview -n tap-install
```

For example:

```
$ tanzu package installed get appliveview -n tap-install

\ Retrieving installation details for appliveview...
NAME:                    appliveview
PACKAGE-NAME:            backend.appliveview.tanzu.vmware.com
PACKAGE-VERSION:         1.5.1
STATUS:                  Reconcile succeeded
```

```
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

Install Application Live View connector

To install Application Live View connector:

1. List version information for the package by running:

   ```
   tanzu package available list connector.appliveview.tanzu.vmware.com --namespace
   tap-install
   ```

   For example:

   ```
   $ tanzu package available list connector.appliveview.tanzu.vmware.com --namespa
   ce tap-install

   - Retrieving package versions for connector.appliveview.tanzu.vmware.com...
     NAME                                    VERSION        RELEASED-AT
     connector.appliveview.tanzu.vmware.com  1.5.1          2023-03-29T00:00:00Z
   ```

2. Create the file `app-live-view-connector-values.yaml` using the following details:

   - **For a single-cluster environment:** The Application Live View connector connects to the `cluster-local` Application Live View back end to register the applications.

     By default, ingress is deactivated for connector. You do not have to change your `app-live-view-connector-values.yaml` file in this step.

   - **For a multicluster environment:** Set the flag `ingressEnabled` to `true` for the Application Live View connector to connect to the Application Live View back end by using the ingress domain. For example:

     ```
     appliveview_connector:
       backend:
         ingressEnabled: true
     ```

   - **For a profile install using shared ingress domain key:** If you are using a Tanzu Application Platform profile installation and the top-level key `shared.ingress_domain` is set in the `tap-values.yaml`, the Application Live View connector and Application Live View back end are configured to communicate through ingress. The Application Live View connector then uses the `shared.ingress_domain` to reach the back end.

     To override the shared ingress for Application Live View in a multicluster environment, use the following values:

     ```
     appliveview_connector:
       backend:
         host: appliveview.INGRESS-DOMAIN
     ```

     Where `INGRESS-DOMAIN` is the top-level domain the Application Live View back end exposes by using `tanzu-shared-ingress` for the connectors in other clusters to reach the Application Live View back end. Prepend the `appliveview` subdomain to the provided value.

3. Configure TLS in your `app-live-view-connector-values.yaml`. Choose a tab depending on how you configured TLS in Install Application Live View back end earlier.

- To configure TLS with a self-signed certificate, set the CA certificate for the ingress domain in the `backend.caCertData` key for SSL validation as follows:

```
appliveview_connector:
  backend:
    ...
    caCertData: |-
      -----BEGIN CERTIFICATE-----
      MIIGMzCCBBugAwIBAgIJALHHzQjxM6wMMA0GCSqGSIb3DQEBDQUAMGcxCzAJBgNV
      BAgMAk1OMRQwEgYDVQQHDAtNaW5uZWFwb2xpczEPMA0GA1UECgwGVk13YXJlMRMw
      -----END CERTIFICATE-----
```

`backend.sslDeactivated` is set to `false` by default.

- To enable TLS using ClusterIssuer:

  1. Retrieve the certificate from the HTTPProxy secret by running the following command in the view cluster:

     ```
     kubectl get secret appliveview-cert -n app-live-view -o yaml |  yq
     '.data."ca.crt"' | base64 -d
     ```

  2. Set the following connector configuration in the run cluster:

     ```
     appliveview_connector:
       backend:
         ingressEnabled: true
         sslDeactivated: false
         host: appliveview.INGRESS-DOMAIN
         caCertData: |-
           -----BEGIN CERTIFICATE-----
           MIIGMzCCBBugAwIBAgIJALHHzQjxM6wMMA0GCSqGSIb3DQEBDQUAMGcxCzAJ
     BgNV
           BAgMAk1OMRQwEgYDVQQHDAtNaW5uZWFwb2xpczEPMA0GA1UECgwGVk13YXJl
     MRMw
           -----END CERTIFICATE-----
     ```

     Where:

     - `caCertData` is the certificate you retrieved from the HTTPProxy secret exposed by the Application Live View back end in the view cluster.
     - `host` is the backend host in the view cluster.

- To deactivate TLS if TLS is not enabled for the `INGRESS-DOMAIN` in the Application Live View back end, set `backend.sslDeactivated` to `true`. For example:

```
appliveview_connector:
  backend:
    ...
    sslDeactivated: true
```

> **Note**
>
> The `sslDisabled` key is deprecated and has been renamed to `sslDeactivated`.

You can edit the values to suit your project needs or leave the default values as is.

Using the HTTP proxy either on 80 or 443 based on SSL config exposes the back end service running on port 7000. The connector connects to the back end on port 80/443 by default. Therefore, you are not required to explicitly configure the `port` field.

4. (Optional) View additional changes you can make in your `app-live-view-connector-values.yaml` file by running:

```
tanzu package available get connector.appliveview.tanzu.vmware.com/VERSION-NUMB
ER \
  --values-schema \
  --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

For example:

```
$ tanzu package available get connector.appliveview.tanzu.vmware.com/1.5.1 \
    --values-schema \
    --namespace tap-install

  KEY                                  DEFAULT             TYPE        DESCRIP
TION
  backend.caCertData                     cert-in-pem-format  string    CA Cert
Data for ingress domain
  backend.host                           <nil>               string    Domain
to be used to reach the application live view backend. Prepend
                                                                       "appliv
eview" subdomain to the value if you are using shared ingress. For
                                                                       exampl
e: "example.com" would become "appliveview.example.com".
  backend.ingressEnabled                 false               boolean   Flag fo
r the connector to connect to ingress on backend

  backend.port                           <nil>               number    Port to
reach the application live view backend
  backend.sslDeactivated                 false               boolean   Flag fo
r whether or not to deactivate ssl
  backend.sslDisabled                    false               boolean   The key
sslDisabled has been deprecated in TAP 1.4.0 and will be removed in TAP
                                                                       1.X+Y.0
of TAP, please migrate to the key sslDeactivated
  connector.namespace_scoped.enabled     false               boolean   Flag fo
r the connector to run in namespace scope
  connector.namespace_scoped.namespace   default             string    Namespa
ce to deploy connector
  kubernetes_distribution                                    string    Kuberne
tes distribution that this package is being installed on. Accepted
                                                                       values:
[''''',''openshift'']
  kubernetes_version                                         string    Optiona
l: The Kubernetes Version. Valid values are '1.24.*', or ''

  activateAppLiveViewSecureAccessControl                     boolean   Optiona
l: Configuration required to enable Secure Access Connection between App
                                                                       Live Vi
ew components
  activateSensitiveOperations                                boolean   Optiona
l: Configuration to allow connector to execute sensitive operations on a
                                                                       running
application
```

5. Install the Application Live View connector package by running:

```
tanzu package install appliveview-connector \
  --package connector.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-live-view-connector-values.yaml
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

For example:

```
$ tanzu package install appliveview-connector \
    --package connector.appliveview.tanzu.vmware.com \
    --version 1.5.1 \
    --namespace tap-install \
    --values-file app-live-view-connector-values.yaml

| Installing package 'connector.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'connector.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-connector-tap-install-sa'
| Creating cluster admin role 'appliveview-connector-tap-install-cluster-role'
| Creating cluster role binding 'appliveview-connector-tap-install-cluster-role
binding'
- Creating package resource
/ Package install status: Reconciling

Added installed package 'appliveview-connector' in namespace 'tap-install'
```

Each cluster installs the connector as a DaemonSet. The connector is configured to connect to the central instance of the back end. The Application Live View connector component is deployed in `app-live-view-connector` namespace by default.

6. Verify the `Application Live View connector` package installation by running:

```
tanzu package installed get appliveview-connector -n tap-install
```

For example:

```
$ tanzu package installed get appliveview-connector -n tap-install

| Retrieving installation details for appliveview-connector...
NAME:                    appliveview-connector
PACKAGE-NAME:            connector.appliveview.tanzu.vmware.com
PACKAGE-VERSION:         1.5.1
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

Install Application Live View conventions

To install Application Live View conventions:

1. List version information for the package by running:

```
tanzu package available list conventions.appliveview.tanzu.vmware.com --namespa
ce tap-install
```

For example:

```
$ tanzu package available list conventions.appliveview.tanzu.vmware.com --names
pace tap-install

- Retrieving package versions for conventions.appliveview.tanzu.vmware.com...
  NAME                                       VERSION        RELEASED-AT
  conventions.appliveview.tanzu.vmware.com  1.5.1          2023-03-29T00:00:00Z
```

2. (Optional) View the changes you can make to the default installation settings by running:

```
tanzu package available get conventions.appliveview.tanzu.vmware.com/VERSION-NU
MBER \
  --values-schema \
  --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

For example:

```
$ tanzu package available get conventions.appliveview.tanzu.vmware.com/1.5.1 \
    --values-schema \
    --namespace tap-install

  KEY                              DEFAULT             TYPE     DESCRIPTION
  kubernetes_distribution                              string  Kubernetes dist
ribution that this package is installed on. Accepted values: ['''',''openshif
t''].
  kubernetes_version                                   string  Optional: The K
ubernetes Version. Valid values are '1.24.*', or ''.
```

3. (Optional) Create a file named `app-live-view-conventions-values.yaml` to override the default installation settings using the information output in the previous step.
4. Install the Application Live View conventions package.

Default values

To install Application Live View conventions with the default settings, run:

```
tanzu package install appliveview-conventions \
  --package conventions.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

For example:

```
$ tanzu package install appliveview-conventions \
    --package conventions.appliveview.tanzu.vmware.com \
    --version 1.5.1 \
    --namespace tap-install

- Installing package 'conventions.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'conventions.appliveview.tanzu.vmware.com'
| Creating service account 'appliveview-conventions-tap-install-sa'
| Creating cluster admin role 'appliveview-conventions-tap-install-cluster-ro
le'
| Creating cluster role binding 'appliveview-conventions-tap-install-cluster-
rolebinding'
- Creating package resource
\ Package install status: Reconciling
```

```
Added installed package 'appliveview-conventions' in namespace 'tap-install'
```

Overriding values

To install Application Live View conventions while overriding the default settings, run:

```
tanzu package install appliveview-conventions \
  --package conventions.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-live-view-conventions-values.yaml
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

5. Verify the package install for Application Live View conventions package by running:

   ```
   tanzu package installed get appliveview-conventions -n tap-install
   ```

   For example:

   ```
   $ tanzu package installed get appliveview-conventions -n tap-install

   | Retrieving installation details for appliveview-conventions...
   NAME:                    appliveview-conventions
   PACKAGE-NAME:            conventions.appliveview.tanzu.vmware.com
   PACKAGE-VERSION:         1.5.1
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

   Verify that `STATUS` is `Reconcile succeeded`.

Install Application Live View APIServer

To install Application Live View APIServer:

1. List version information for the package by running:

   ```
   tanzu package available list apiserver.appliveview.tanzu.vmware.com --namespace
   tap-install
   ```

   For example:

   ```
   $ tanzu package available list apiserver.appliveview.tanzu.vmware.com --namespa
   ce tap-install

   - Retrieving package versions for apiserver.appliveview.tanzu.vmware.com...
     NAME                                    VERSION       RELEASED-AT
     apiserver.appliveview.tanzu.vmware.com  1.5.1         2023-03-29T00:00:00Z
   ```

2. (Optional) View the changes you can make to the default installation settings by running:

   ```
   tanzu package available get apiserver.appliveview.tanzu.vmware.com/VERSION-NUMB
   ER \
     --values-schema \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

   For example:

```
$ tanzu package available get apiserver.appliveview.tanzu.vmware.com/1.5.1 \
    --values-schema \
    --namespace tap-install \

  KEY                                DEFAULT             TYPE     DESCRIPTION
  kubernetes_distribution                                string  Kubernetes dist
ribution that this package is installed on. Accepted values: ['''','openshif
t''].
  kubernetes_version                                     string  Optional: The K
ubernetes Version. Valid values are '1.24.*', or ''.
```

3. (Optional) Create a file named `app-live-view-apiserver-values.yaml` to override the default installation settings using the information output in the previous step.

4. Install the Application Live View APIServer package.

Default values

To install Application Live View APIServer with the default settings, run:

```
tanzu package install appliveview-apiserver \
  --package apiserver.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

For example:

```
$ tanzu package install appliveview-apiserver \
    --package apiserver.appliveview.tanzu.vmware.com \
    --version 1.5.1 \
    --namespace tap-install

- Installing package 'apiserver.appliveview.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'apiserver.appliveview.tanzu.vmware.com'
- Creating package resource
\ Package install status: Reconciling

Added installed package 'appliveview-apiserver' in namespace 'tap-install'
```

Overriding values

To install Application Live View APIServer while overriding the default settings, run:

```
tanzu package install appliveview-apiserver \
  --package apiserver.appliveview.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-live-view-apiserver-values.yaml
```

Where `VERSION-NUMBER` is the version of the package listed earlier. For example, `1.5.1`.

5. Verify the package install for Application Live View APIServer package by running:

```
tanzu package installed get appliveview-apiserver -n tap-install
```

For example:

```
$ tanzu package installed get appliveview-apiserver -n tap-install

| Retrieving installation details for appliveview-apiserver...
```

```
NAME:                     appliveview-apiserver
PACKAGE-NAME:             apiserver.appliveview.tanzu.vmware.com
PACKAGE-VERSION:          1.5.1
STATUS:                   Reconcile succeeded
CONDITIONS:               [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

The Application Live View UI plug-in is part of Tanzu Application Platform GUI. To access the Application Live View UI, see Application Live View in Tanzu Application Platform GUI.

Deprecation notice for the sslDisabled key

The `appliveview_connector.backend.sslDisabled` key is deprecated and renamed to `appliveview_connector.backend.sslDeactivated`. The `appliveview_connector.backend.sslDisabled` key is marked for removal in Tanzu Application Platform 1.7.0.

Configure security and access control in Application Live View

This topic tells you how to enable improved security and access control in Application Live View in Tanzu Application Platform (commonly known as TAP). Improved security and access control in Application Live View secures the REST API exposed by the Application Live View back end.

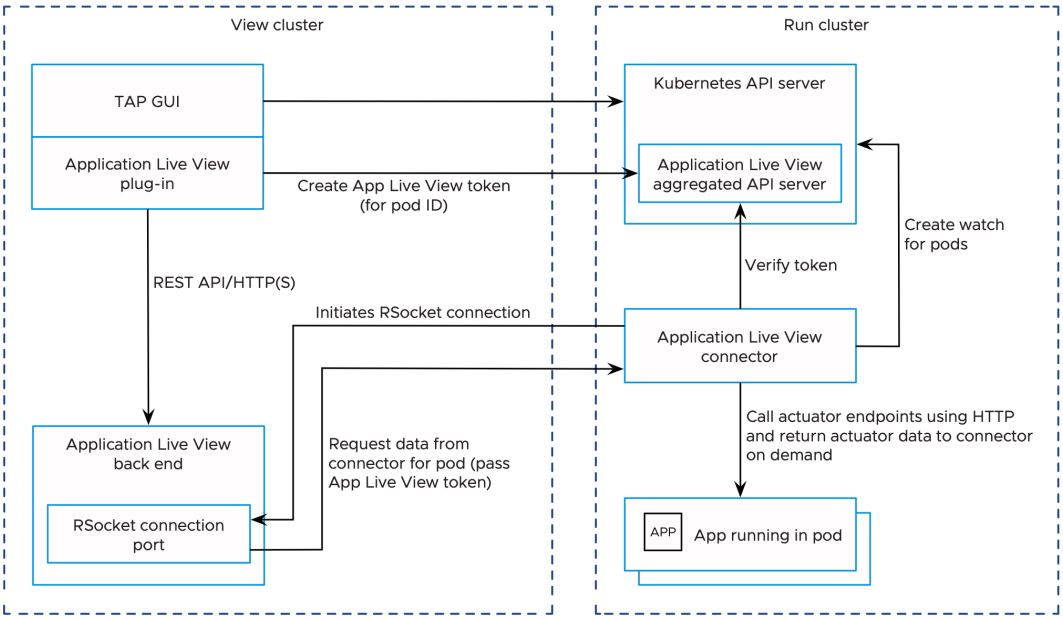
For more information about Application Live View packages, see Install Application Live View.

Security and access control overview

There is one instance of Application Live View back end installed per View profile. Multiple users access this back-end API to fetch actuator data for different applications. All the REST API calls to the back end are secured. A token must be passed to the Application Live View back end on each call to the REST API to fetch actuator data. This token is obtained from Application Live View APIServer.

The Application Live View APIServer generates a unique token upon access validation of a user to a pod. The Application Live View back end passes this token to the Application Live View connector when requesting the actuator data. The Application Live View connector verifies this token by calling the Application Live View APIServer and proxies the actuator data only if the token is valid.

The Application Live View UI plug-in part of The Tanzu Application Platform GUI uses the preceding approach to securely query for the actuator data for a pod. It requests a token from Application Live View APIServer and passes it in the subsequent calls to the back end. This ensures that actuator data from the running application is fetched only if the user is authorized to see the live information for the pod.



The Application Live View UI plug-in relies on Tanzu Application Platform GUI authentication and authorization to access the Application Live View APIServer and fetch the Application Live View tokens.

The Tanzu Application Platform GUI controls the access to Kubernetes resources based on user roles and permissions for each of the remote clusters. For more information, see View runtime resources on authorization-enabled clusters.

For more information about how to set up unrestricted remote cluster visibility, see Viewing resources on multiple clusters in Tanzu Application Platform GUI.

Prerequisites

1. You install the Application Live View APIServer package (`apiserver.appliveview.tanzu.vmware.com`) in Tanzu Application Platform. For more information, see Install Application Live View APIServer.
2. Assign users necessary roles and permissions for the Kubernetes clusters. For more information about managing role-based access control, see Assign roles and permissions on Kubernetes clusters

For example: If you are using Service Account to view resources on a cluster in Tanzu Application Platform GUI, verify that the `ClusterRole` has rules to access and request tokens from the Application Live View APIServer.

```
- apiGroups: ['appliveview.apps.tanzu.vmware.com']
  resources:
  - resourceinspectiongrants
  verbs: ['get', 'watch', 'list', 'create']
```

For more information, see Set up a Service Account to view resources on a cluster.

Note

With the Service Account approach to view resources on a cluster, every user with the Service Account Token with access to view the pods is able to see the live information in Application Live View.

Configure improved security

The improved security feature is enabled by default for Application Live View.

In a Tanzu Application Platform profile install, the Application Live View connector (`connector.appliveview.tanzu.vmware.com`) and the Tanzu Application Platform GUI (`tap-gui.tanzu.vmware.com`) are automatically configured to enable the secure communication between Application Live View components.

You can control this feature by setting the top-level key `shared.activateAppLiveViewSecureAccessControl` in the `tap-values.yaml`.

For example:

```
shared:
    activateAppLiveViewSecureAccessControl: true
```

To override the security feature at the individual component level, take the following steps.

Application Live View connector

1. (Optional) Change the default installation settings for Application Live View connector by running:

   ```
   tanzu package available get connector.appliveview.tanzu.vmware.com/VERSION-NUMB
   ER --values-schema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the package listed. For example, `1.5.0-build.5`.

   For example:

   ```
   $ tanzu package available get connector.appliveview.tanzu.vmware.com/1.5.0-buil
   d.5 --values-schema --namespace tap-install
     KEY                                 DEFAULT             TYPE        DESCRIP
   TION
     kubernetes_version                                      string      Optiona
   l: The Kubernetes Version. Valid values are '1.24.*', or ''.

     backend.sslDeactivated              false               boolean     Flag fo
   r whether to disable SSL.
     backend.caCertData                  cert-in-pem-format  string      CA Cert
   Data for ingress domain.
     backend.host                        <nil>               string      Domain
   used to reach the Application Live View back end. Prepend
                                                                           "appliv
   eview" subdomain to the value if you use shared ingress. For
                                                                           exampl
   e: "example.com" becomes "appliveview.example.com".
     backend.ingressEnabled              false               boolean     Flag fo
   r the connector to connect to ingress on back end.

     backend.port                        <nil>               number      Port to
   reach the Application Live View back end.
     connector.namespace_scoped.enabled  false               boolean     Flag fo
   r the connector to run in namespace scope.
     connector.namespace_scoped.namespace  default           string      Namespa
   ce to deploy connector.
   ```

```
  kubernetes_distribution                                    string      Kuberne
tes distribution that this package is being installed on. Accepted
                                                                         values:
[''''],''openshift''].
  activateAppLiveViewSecureAccessControl                     boolean     Optiona
l: Configuration required to enable Secure Access Connection between App Live V
iew components.
  activateSensitiveOperations                                boolean     Optiona
l: Configuration to allow connector to execute sensitive operations on a runnin
g application.
```

For more information about values schema options, see the properties listed earlier.

2. Create `app-live-view-connector-values.yaml` with the following details:

   To override the default security settings for connector, use the following values:

   ```
   activateAppLiveViewSecureAccessControl: false
   ```

   By default, activateAppLiveViewSecureAccessControl is set to `true`.

   The `activateSensitiveOperations` key activates/deactivates the execution of sensitive operations, such as editing environment variables, downloading heap dump data, and changing log levels for the applications in the cluster. It is set to `false` by default.

   To enable the sensitive operations, set `activateSensitiveOperations` to `true`.

   ```
   activateSensitiveOperations: true
   ```

3. Install the Application Live View connector package by running:

   ```
   tanzu package install appliveview-connector \
     --package connector.appliveview.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install \
     --values-file app-live-view-connector-values.yaml
   ```

   Where `VERSION-NUMBER` is the version of the package listed. For example, `1.5.0-build.5`.

   For example:

   ```
   $ tanzu package install appliveview-connector \
       --package connector.appliveview.tanzu.vmware.com \
       --version 1.5.0-build.5 \
       --namespace tap-install \
       --values-file app-live-view-connector-values.yaml
   | Installing package 'connector.appliveview.tanzu.vmware.com'
   | Getting namespace 'tap-install'
   | Getting package metadata for 'connector.appliveview.tanzu.vmware.com'
   | Creating service account 'appliveview-connector-tap-install-sa'
   | Creating cluster admin role 'appliveview-connector-tap-install-cluster-role'
   | Creating cluster role binding 'appliveview-connector-tap-install-cluster-role
   binding'
   - Creating package resource
   / Package install status: Reconciling

   Added installed package 'appliveview-connector' in namespace 'tap-install'
   ```

4. Verify the `Application Live View connector` package installation by running:

   ```
   tanzu package installed get appliveview-connector -n tap-install
   ```

   For example:

```
tanzu package installed get appliveview-connector -n tap-install
5s
| Retrieving installation details for appliveview-connector...
NAME:                    appliveview-connector
PACKAGE-NAME:            connector.appliveview.tanzu.vmware.com
PACKAGE-VERSION:         1.5.0-build.5
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

Application Live View UI plug-in

The Application Live View UI plug-in is part of Tanzu Application Platform GUI. To override the default security settings for the Application Live View UI plug-in, take the following steps.

1. (Optional) Make changes to the default installation settings by running:

   ```
   tanzu package available get tap-gui.tanzu.vmware.com/VERSION-NUMBER --values-sc
   hema --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the number you discovered previously. For example, `1.4.6`.

   For more information about values schema options, see the individual product documentation.

2. Create `tap-gui-values.yaml` and paste in the following code:

   ```
   ingressEnabled: true
   ingressDomain: "INGRESS-DOMAIN"
   app_config:
     catalog:
       locations:
         - type: url
           target: https://GIT-CATALOG-URL/catalog-info.yaml
     appLiveView:
       activateAppLiveViewSecureAccessControl: false
   ```

   Where:

   - `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
   - `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file from either the included Blank catalog (provided as an additional download named "Blank Tanzu Application Platform GUI Catalog") or a Backstage-compliant catalog that you've already built and posted on the Git infrastructure specified in Add Tanzu Application Platform GUI integrations.

3. Install the package by running:

   ```
   tanzu package install tap-gui \
     --package tap-gui.tanzu.vmware.com \
     --version VERSION \
     --namespace tap-install \
     --values-file tap-gui-values.yaml
   ```

   Where `VERSION` is the version that you want. For example, `1.4.6`.

   For example:

```
$ tanzu package install tap-gui \
    --package tap-gui.tanzu.vmware.com \
    --version 1.4.6 \
    --namespace tap-install \
    --values-file tap-gui-values.yaml
- Installing package 'tap-gui.tanzu.vmware.com'
| Getting package metadata for 'tap-gui.tanzu.vmware.com'
| Creating service account 'tap-gui-default-sa'
| Creating cluster admin role 'tap-gui-default-cluster-role'
| Creating cluster role binding 'tap-gui-default-cluster-rolebinding'
| Creating secret 'tap-gui-default-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'tap-gui' in namespace 'tap-install'
```

4. Verify that the package installed by running:

   ```
   tanzu package installed get tap-gui -n tap-install
   ```

   For example:

   ```
   $ tanzu package installed get tap-gui -n tap-install
   | Retrieving installation details for cc...
   NAME:                    tap-gui
   PACKAGE-NAME:            tap-gui.tanzu.vmware.com
   PACKAGE-VERSION:         1.4.6
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

   Verify that `STATUS` is `Reconcile succeeded`.

5. To access Tanzu Application Platform GUI, use the service you exposed in the `service_type` field in the values file.

Enabling Spring Boot apps for Application Live View

This topic for developers tells you how to configure a Spring Boot app for observation by Application Live View within Tanzu Application Platform (commonly known as TAP).

Enable Spring Boot apps

For Application Live View to interact with a Spring Boot app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add the following plugin configuration in `pom.xml`:

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
```

```
        <goal>build-info</goal>
      </goals>
      <configuration>
        <additionalProperties>
          <spring.boot.version>${project.parent.version}</spring.boot.version>
        </additionalProperties>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Add the preceding configuration to generate `build-info.properties` into your Spring Boot application. This information is then used to display the Spring Boot version that the app uses in Application Live View.

To enable Application Live View for Spring Boot apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information about the labels automatically set by Spring Boot conventions, see Enable Application Live View for Spring Boot applications.

Enable Spring Boot 3 apps

For Application Live View to interact with a Spring Boot 3 app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add the following plugin configuration in `pom.xml`:

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>build-info</goal>
      </goals>
      <configuration>
        <additionalProperties>
          <spring.boot.version>${project.parent.version}</spring.boot.version>
        </additionalProperties>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Add the preceding configuration to generate `build-info.properties` into your Spring Boot application. This information is then used to display the Spring Boot version that the app uses in Application Live View.

To enable Application Live View for Spring Boot 3 apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information about the labels automatically set by Spring Boot conventions, see Enable Application Live View for Spring Boot applications.

Here is an example of creating a workload for a Spring Boot 3 Application:

```
tanzu apps workload create spring-boot-3 --git-repo https://github.com/martinlippert/sb3-demo.git --git-branch main --annotation autoscaling.knative.dev/min-scale=1 --yes --label app.kubernetes.io/part-of=tanzu-java-web-app --type web --build-env "BP_JVM_VERSION=17" --label apps.tanzu.vmware.com/auto-configure-actuators="true"
```

Enable Spring Cloud Gateway apps

For Application Live View to interact with a Spring Cloud Gateway app within Tanzu Application Platform, add the `spring-boot-starter-actuator` and `spring-cloud-starter-gateway` module dependency.

Add the maven dependencies in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

To enable Application Live View on the Spring Cloud Gateway Tanzu Application Platform workload, Spring Boot conventions automatically applies labels on the workload, such as `tanzu.app.live.view.application.flavours: spring-boot_spring-cloud-gateway` and `tanzu.app.live.view: true`, based on the Spring Cloud Gateway image metadata.

Here is an example of creating a workload for a Spring Cloud Gateway Application:

```
tanzu apps workload create tanzu-scg-web-app --git-repo https://github.com/ksankaranara-vmw/gs-gateway.git --git-branch main --type web --label app.kubernetes.io/part-of=tanzu-scg-web-app --yes --annotation autoscaling.knative.dev/min-scale=1
```

Workload image NOT built with Tanzu Build Service

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Boot` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create boot-app --type web --app boot-app --image <IMAGE NAME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --label tanzu.app.live.view.application.name=boot-app --label tanzu.app.live.view.application.flavours=spring-boot
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Cloud Gateway` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create scg-app --type web --app scg-app --image <IMAGE NAME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --label tanzu.app.live.view.application.name=scg-app --label tanzu.app.live.view.application.flavours=spring-boot_spring-cloud-gateway
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Steeltoe` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create steeltoe-app --type web --app steeltoe-app --image <IMAGE NAME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.vie
```

```
w=true --label tanzu.app.live.view.application.name=steeltoe-app --label tanzu.app.liv
e.view.application.flavours=steeltoe
```

Enabling Spring Boot apps for Application Live View

This topic for developers tells you how to configure a Spring Boot app for observation by Application Live View within Tanzu Application Platform (commonly known as TAP).

Enable Spring Boot apps

For Application Live View to interact with a Spring Boot app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add the following plugin configuration in `pom.xml`:

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>build-info</goal>
      </goals>
      <configuration>
        <additionalProperties>
          <spring.boot.version>${project.parent.version}</spring.boot.version>
        </additionalProperties>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Add the preceding configuration to generate `build-info.properties` into your Spring Boot application. This information is then used to display the Spring Boot version that the app uses in Application Live View.

To enable Application Live View for Spring Boot apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information about the labels automatically set by Spring Boot conventions, see Enable Application Live View for Spring Boot applications.

Enable Spring Boot 3 apps

For Application Live View to interact with a Spring Boot 3 app within Tanzu Application Platform, add the `spring-boot-starter-actuator` module dependency.

Add the maven dependency in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Add the following plugin configuration in `pom.xml`:

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>build-info</goal>
      </goals>
      <configuration>
        <additionalProperties>
          <spring.boot.version>${project.parent.version}</spring.boot.version>
        </additionalProperties>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Add the preceding configuration to generate `build-info.properties` into your Spring Boot application. This information is then used to display the Spring Boot version that the app uses in Application Live View.

To enable Application Live View for Spring Boot 3 apps, Spring Boot conventions automatically sets the Application Live View labels onto the PodSpec. For more information about the labels automatically set by Spring Boot conventions, see Enable Application Live View for Spring Boot applications.

Here is an example of creating a workload for a Spring Boot 3 Application:

```
tanzu apps workload create spring-boot-3 --git-repo https://github.com/martinlippert/s
b3-demo.git --git-branch main --annotation autoscaling.knative.dev/min-scale=1 --yes -
-label app.kubernetes.io/part-of=tanzu-java-web-app --type web --build-env "BP_JVM_VER
SION=17" --label apps.tanzu.vmware.com/auto-configure-actuators="true"
```

Enable Spring Cloud Gateway apps

For Application Live View to interact with a Spring Cloud Gateway app within Tanzu Application Platform, add the `spring-boot-starter-actuator` and `spring-cloud-starter-gateway` module dependency.

Add the maven dependencies in `pom.xml` as follows:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

To enable Application Live View on the Spring Cloud Gateway Tanzu Application Platform workload, Spring Boot conventions automatically applies labels on the workload, such as `tanzu.app.live.view.application.flavours: spring-boot_spring-cloud-gateway` and `tanzu.app.live.view: true`, based on the Spring Cloud Gateway image metadata.

Here is an example of creating a workload for a Spring Cloud Gateway Application:

```
tanzu apps workload create tanzu-scg-web-app --git-repo https://github.com/ksankaranar
a-vmw/gs-gateway.git --git-branch main --type web --label app.kubernetes.io/part-of=ta
```

```
nzu-scg-web-app --yes --annotation autoscaling.knative.dev/min-scale=1
```

Workload image NOT built with Tanzu Build Service

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Boot` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create boot-app --type web --app boot-app --image <IMAGE NAME> --a
nnotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true -
-label tanzu.app.live.view.application.name=boot-app --label tanzu.app.live.view.appli
cation.flavours=spring-boot
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Spring Cloud Gateway` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create scg-app --type web --app scg-app --image <IMAGE NAME> --ann
otation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=true --l
abel tanzu.app.live.view.application.name=scg-app --label tanzu.app.live.view.applicat
ion.flavours=spring-boot_spring-cloud-gateway
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on `Steeltoe` Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create steeltoe-app --type web --app steeltoe-app --image <IMAGE N
AME> --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.vie
w=true --label tanzu.app.live.view.application.name=steeltoe-app --label tanzu.app.liv
e.view.application.flavours=steeltoe
```

Enable Steeltoe apps for Application Live View

This topic for developers tells you how to extend .NET Core Apps to Steeltoe apps and enable Application Live View on Steeltoe workloads within Tanzu Application Platform (commonly known as TAP).

Application Live View supports Steeltoe .NET apps with .NET core runtime version `v6.0.8`.

Extend .NET Core Apps to Steeltoe Apps

A .NET Core application can be extended to a Steeltoe application by adding independent NuGet packages.

To enable the Actuators on a .NET Core App:

1. Add a PackageReference to your `.csproj` file:

   ```
   <PackageReference Include="Steeltoe.Management.EndpointCore" Version="$(Steelto
   eVersion)" />
   ```

   Note

   The PackageReference is expected to change to `Steeltoe.Management.Endpoint` from version Steeltoe 4.0 onwards.

2. Call the extension `AddAllActuators` in your `Program.cs` file:

```
builder.WebHost.AddAllActuators();
```

3. (Optional) You can add app-specific configurations, such as the following.

   To expose all management actuator endpoints except `env` endpoint, add the following configuration to your `appsettings.json` file:

```
{
  "Management": {
    "Endpoints": {
      "Actuator":{
        "Exposure": {
          "Include": [ "*" ],
          "Exclude": [ "env" ]
        }
      }
    }
  }
}
```

   To enable logging, add the following configuration to your `appsettings.json` file:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Steeltoe": "Warning",
      "Sample": "Information"
    }
  }
}
```

   To enable heapdump, add the following configuration to your `appsettings.json` file:

```
{
  "Management": {
    "Endpoints": {
      "HeapDump": {
        "HeapDumpType": "Normal"
      }
    }
  }
}
```

Enable Application Live View on Steeltoe Tanzu Application Platform workload

You can enable Application Live View to interact with a Steeltoe app within Tanzu Application Platform.

To enable Application Live View on the Steeltoe Tanzu Application Platform workload, the Application Live View convention service automatically applies labels on the workload, such as `tanzu.app.live.view.application.flavours: steeltoe` and `tanzu.app.live.view: true`, based on the Steeltoe image metadata.

Here's an example of creating a workload for a Steeltoe Application:

```
tanzu apps workload create steeltoe-app --type web --git-repo https://github.com/vmwar
e-tanzu/application-accelerator-samples --sub-path weatherforecast-steeltoe --git-bran
```

```
ch main --annotation autoscaling.knative.dev/min-scale=1 --yes --label app.kubernetes.
io/part-of=sample-app
```

If your application image is NOT built with Tanzu Build Service, to enable Application Live View on Steeltoe Tanzu Application Platform workload, use the following command. For example:

```
tanzu apps workload create steeltoe-app --type web --app steeltoe-app --image IMAGE-NA
ME --annotation autoscaling.knative.dev/min-scale=1 --yes --label tanzu.app.live.view=
true --label tanzu.app.live.view.application.name=steeltoe-app --label tanzu.app.live.
view.application.flavours=steeltoe
```

Where `IMAGE-NAME` is the name of your application image.

Note

Thread metrics is available in SteeltoeVersion `3.2.*`. To enable the Threads page in the Application Live View UI, add the following configuration to your `.csproj` file:

```
<PropertyGroup>
    <SteeltoeVersion>3.2.*</SteeltoeVersion>
</PropertyGroup>
```

Application Live View convention server

This topic provides information about Application Live View convention, which provides a Webhook handler for Cartographer Conventions.

Role of Application Live View convention

Application Live View conventions works in conjunction with core Cartographer Conventions. It enhances Tanzu PodIntents with metadata such as labels, annotations, or app properties. This metadata allows Application Live View, specifically the connector, to discover app instances so that Application Live View can access the actuator data from those workloads.

Note

Application Live View conventions now supports only Steeltoe applications. Spring Boot conventions supports both Spring Boot and Spring Cloud Gateway applications. For more information about Spring Boot conventions, see Enable Application Live View with Spring Boot apps

To run Application Live View with Steeltoe apps, the Spring Boot convention recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Enables the connector to observe application pod.
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View.
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View.
- `tanzu.app.live.view.application.flavours: steeltoe`: Exposes the framework flavor of the app.

These metadata labels allow Application Live View to identify pods that are enabled for Application Live View. The metadata labels also tell the Application Live View connector what kind of app it is, and on which port the actuators are accessible for Application Live View.

Description of metadata labels

If a workload resource explicitly defines a label under `metadata.labels` in the `workload.yaml`, then the convention service detects the presence of that label and respects its value. When deploying a workload using Tanzu Application Platform, you can override the labels listed in the following table using the `Workload` YAML.

Metadata	Default	Type	Description
`tanzu.app.live.view`	`true`	Label	When deploying a workload in Tanzu Application Platform, this label is set to `true` as default across the supply chain.
`tanzu.app.live.view.application.name`	`steeltoe-app`	Label	When deploying a workload in Tanzu Application Platform, this label is set to `steeltoe-app` if the container image metadata does not contain the app name. Otherwise, the label is set to the app name from container image metadata.
`tanzu.app.live.view.application.flavours`	`steeltoe`	Label	When deploying a Spring Boot workload in Tanzu Application Platform, this label is set to `steeltoe` as default across the supply chain.

Verify the applied labels and annotations

You can verify the applied labels and annotations by running:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -o yaml
```

Where `WORKLOAD-NAME` is the name of the deployed workload, for example `steetoe-app`.

Expected output for Steeltoe workload:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2022-11-14T09:56:53Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: sample-app
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: steeltoe-app
    carto.run/workload-namespace: default
  name: steeltoe-app
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: steeltoe-app
    uid: 97897399-807a-4815-9693-fb06bb4bc1ed
  resourceVersion: "428904"
  uid: 0c74e045-075c-4af3-beef-b092b951be7f
spec:
```

```
  serviceAccountName: default
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: sample-app
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: steeltoe-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/steeltoe-app-default@s
ha256:c8ea14d8714ec31ad978085ebff43d15679613a0c12df37812adf22cb47b5232
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2022-11-14T09:56:57Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2022-11-14T09:56:57Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/auto-configure-actuators-check
          spring-boot-convention/app-live-view-appflavour-check
          appliveview-sample/app-live-view-appflavour-check
          appliveview-sample/app-live-view-connector-steeltoe
          appliveview-sample/app-live-view-appflavours-steeltoe
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: sample-app
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: steeltoe-app
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.flavours: steeltoe
        tanzu.app.live.view.application.name: steeltoe-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/steeltoe-app-default@s
ha256:c8ea14d8714ec31ad978085ebff43d15679613a0c12df37812adf22cb47b5232
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
```

In your output:

- `status.template.metadata.labels` shows the list of applied labels by Application Live View convention server.

- `status.template.metadata.annotations` shows the list of applied annotations by Application Live View convention server.

Custom configuration for the connector

This topic for developers tells you how to custom configure an app or workload for Application Live View.

The connector component is responsible for discovering the app and registering it with Application Live View. Labels from the app PodSpec are used to discover the app and configure the connector to access the actuator data of the app.

Usually, Application Live View conventions applies the necessary configuration automatically. To deactivate the convention and configure the app and the workload manually, the list of labels in the following table gives you an overview of the options:

Label Name	Mandatory	Type	Default	Significance
`tanzu.app.live.view`	true	Boolean	None	Toggle to activate or deactivate pod discovery
`tanzu.app.live.view.application.name`	true	String	None	Application name
`tanzu.app.live.view.application.port`	false	Integer	`8080`	Application port
`tanzu.app.live.view.application.path`	false	String	`/`	Application context path
`tanzu.app.live.view.application.actuator.port`	false	Integer	`8080`	Application actuator port
`tanzu.app.live.view.application.actuator.path`	false	String	`/actuator`	Actuator context path
`tanzu.app.live.view.application.protocol`	false	http / https	`http`	Protocol scheme
`tanzu.app.live.view.application.actuator.health.port`	false	Integer	`8080`	Health endpoint port
`tanzu.app.live.view.application.flavours`	false	Comma separated string	`spring-boot,spring-cloud-gateway`	Application flavors

You can add connector labels in the app `Workload` or override labels that the convention applies, such as `tanzu.app.live.view` and `tanzu.app.live.view.application.name`. If you do not want Application Live View to observe your app, you can override the existing label `tanzu.app.live.view: "false"`.

Configure the developer workload in Tanzu Application Platform

The following YAML is an example of a Spring PetClinic workload that overrides the connector label to `tanzu.app.live.view: "false"`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
  namespace: default
  labels:
```

```
    tanzu.app.live.view: "false"
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  annotations:
    autoscaling.knative.dev/minScale: "1"
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/spring-petclinic
```

Deploy the workload

To deploy the workload, run:

```
kapp -y deploy -n default -a workloads -f workloads.yaml
```

Verify the label has propagated through the Supply Chain

To verify the label:

1. Verify that the workload build is successful by ensuring that `SUCCEEDED` is set to `True`:

   ```
   kubectl get builds
   NAME                          IMAGE
   SUCCEEDED
   spring-petclinic-build-1      dev.registry.tanzu.vmware.com/app-live-view/test/s
   pring-petclinic-default@sha256:9db2a8a8e77e9215239431fd8afe3f2ecdf09cce8afac565
   dad7b5f0c5ac0cdf     True
   ```

2. Verify the PodIntent of your workload by ensuring `status.template.metadata.labels` shows the newly added label has propagated through the Supply Chain:

   ```
   kubectl get podintents.conventions.carto.run spring-petclinic -oyaml

   status:
     conditions:
     - lastTransitionTime: "2021-12-03T15:14:33Z"
       status: "True"
       type: ConventionsApplied
     - lastTransitionTime: "2021-12-03T15:14:33Z"
       status: "True"
       type: Ready
     observedGeneration: 3
     template:
       metadata:
         annotations:
           autoscaling.knative.dev/minScale: "1"
           boot.spring.io/actuator: http://:8080/actuator
           boot.spring.io/version: 2.5.6
           conventions.carto.run/applied-conventions: |-
             appliveview-sample/app-live-view-connector-boot
             appliveview-sample/app-live-view-appflavours-boot
             appliveview-sample/app-live-view-systemproperties
             spring-boot-convention/spring-boot
             spring-boot-convention/spring-boot-graceful-shutdown
             spring-boot-convention/spring-boot-web
             spring-boot-convention/spring-boot-actuator
             spring-boot-convention/service-intent-mysql
           developer.conventions/target-containers: workload
           kapp.k14s.io/identity: v1;default/carto.run/Workload/spring-petclinic;c
   ```

```
arto.run/v1alpha1
        kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Work
load","metadata":{"annotations":{"autoscaling.knative.dev/minScale":"2"},"label
s":{"app.kubernetes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/wor
kload-type":"web","kapp.k14s.io/app":"1638455805474051000","kapp.k14s.io/associ
ation":"v1.5a9384bd7b93ca74ef494c4dec2caa4b","tanzu.app.live.view":"false"},"na
me":"spring-petclinic","namespace":"default"},"spec":{"source":{"git":{"ref":
{"branch":"main"},"url":"https://github.com/ksankaranara-vmw/spring-petclini
c"}}}}'
        kapp.k14s.io/original-diff-md5: 58e0494c51d30eb3494f7c9198986bb9
        services.conventions.carto.run/mysql: mysql-connector-java/8.0.27
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: spring-petclinic
        conventions.carto.run/framework: spring-boot
        kapp.k14s.io/app: "1638455805474051000"
        kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
        services.conventions.carto.run/mysql: workload
        tanzu.app.live.view: "false"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: petclinic
```

3. Verify the ConfigMap was created for the app by ensuring `metadata.labels` shows the newly added label has propagated through the Supply Chain:

```
kubectl describe configmap spring-petclinic
Name:         spring-petclinic
Namespace:    default
Labels:       carto.run/cluster-supply-chain-name=source-to-url
              carto.run/cluster-template-name=config-template
              carto.run/resource-name=app-config
              carto.run/template-kind=ClusterConfigTemplate
              carto.run/workload-name=spring-petclinic
              carto.run/workload-namespace=default
Annotations:  <none>

Data
====
delivery.yml:
----
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    kapp.k14s.io/app: "1638455805474051000"
    kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
    tanzu.app.live.view: "false"
    app.kubernetes.io/component: run
    carto.run/workload-name: spring-petclinic
```

4. Verify the running Knative application pod by ensuring `labels` shows the newly added label on the Knative application pod:

```
kubectl get pods -o yaml spring-petclinic-00002-deployment-77dbb85c65-cf7rn | g
rep labels
    kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workloa
d","metadata":{"annotations":{"autoscaling.knative.dev/minScale":"1"},"labels":
{"app.kubernetes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/worklo
ad-type":"web","kapp.k14s.io/app":"1638455805474051000","kapp.k14s.io/associati
on":"v1.5a9384bd7b93ca74ef494c4dec2caa4b","tanzu.app.live.view":"false"},"nam
```

```
e":"spring-petclinic","namespace":"default"},"spec":{"source":{"git":{"ref":{"b
ranch":"main"},"url":"https://github.com/ksankaranara-vmw/spring-petclini
c"}}}}'
```

You can add or override the connector in the `Workload` of your Knative app.

Custom configuration for application actuator endpoints

This topic for developers tells you how to configure the Application Live View connector component to access actuator endpoints for custom settings, such as a different base path. By default, the actuator endpoint for an application is exposed on `/actuator`.

The following table describes the actuator configuration scenarios and the associated labels to use, assuming that the app runs on port `8080`:

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
None	None	None	None	Actuators endpoints available at `localhost:8080/actuator`	`tanzu.app.live.view.application.actuator.path=actuator`, `tanzu.app.live.view.application.actuator.port=8080`
`/path`	`8082`	`/`	None	Actuator endpoints available at `localhost:8082/path`	`tanzu.app.live.view.application.actuator.path=path`, `tanzu.app.live.view.application.actuator.port=8082`
`/path`	`8082`	`/manage/actuator`	None	Actuator endpoints available at `localhost:8082/path/manage/actuator`	`tanzu.app.live.view.application.actuator.path=path/manage/actuator`, `tanzu.app.live.view.application.actuator.port=8082`
None	None	`/`	None	Actuators are deactivated to avoid conflicts	None

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
None	None	`/manage`	None	Actuator endpoints available at `/manage`	`tanzu.app.live.view.application.actuator.path=manage`, `tanzu.app.live.view.application.actuator.port=8080`
`/path`	`8082`	None	None	Actuator endpoints available at `localhost:8082/path/actuator`	`tanzu.app.live.view.application.actuator.path=path/actuator`, `tanzu.app.live.view.application.actuator.port=8082`
`/`	`8082`	None	None	Actuator endpoints available at `localhost:8082/actuator`	`tanzu.app.live.view.application.actuator.path=actuator`, `tanzu.app.live.view.application.actuator.port=8082`
None	None	None	`/api`	Actuator endpoints available at `localhost:8080/api/actuator`	`tanzu.app.live.view.application.actuator.path=api/actuator`, `tanzu.app.live.view.application.actuator.port=8080`
`/path`	`8082`	None	`/api`	Actuator endpoints available at `localhost:8082/path/actuator`	`tanzu.app.live.view.application.actuator.path=path/actuator`, `tanzu.app.live.view.application.actuator.port=8082`

management.server.base-path	management.server.port	management.endpoints.web.base-path	server.servlet.context.path	Comments	Connector Configuration
`/path`	`8082`	`/manage`	`/api`	Actuator endpoints available at `localhost:8082/path/manage`	`tanzu.app.live.view.application.actuator.path=path/manage`, `tanzu.app.live.view.application.actuator.port=8082`
`/path`	None	`/manage`	`/api`	Actuator endpoints available at `localhost:8080/api/manage`	`tanzu.app.live.view.application.actuator.path=api/manage`, `tanzu.app.live.view.application.actuator.port=8080`
`/path`	None	`/`	`/api`	Actuators are deactivated to avoid conflicts	None
`/path`	`8082`	`/`	`/api`	Actuator endpoints available at `localhost:8082/path`	`tanzu.app.live.view.application.actuator.path=path`, `tanzu.app.live.view.application.actuator.port=8082`
None	None	`/manage`	`/api`	Actuator endpoints available at `localhost:8080/api/manage`	`tanzu.app.live.view.application.actuator.path=api/manage`, `tanzu.app.live.view.application.actuator.port=8080`

Scaling Knative apps in Tanzu Application Platform

This topic tells you how to use Application Live View when scaling Knative apps or developer workloads in Tanzu Application Platform (commonly known as TAP).

Application Live View is focused on monitoring apps for a `live` window and does not apply to any of the apps that are scaled down to zero. The intended behavior for Knative apps is to keep track of revisions to allow you to rollback easily, but also scale all of the unused revision instances down to zero to keep resource consumption low.

You can configure Knative apps to set `autoscaling.knative.dev/minScale` to `1` so that Application Live View can still observe app instance. This ensures that there is at least one instance of the latest revision, while still scaling down the older instances.

You can configure any app in Tanzu Application Platform using the `Workload` resource. To scale a Knative app, add the annotation `autoscaling.knative.dev/minScale` in the `Workload` and set it to the value you want. For Application Live View to observe an app and have at least one instance of the latest revision, set `autoscaling.knative.dev/minScale = "1"`.

The annotations or labels in the `Workload` get propagated through the Tanzu Application Platform supply chain as follows:

Workload > PodIntent > ConfigMap > Push Config > to repository/registry > git-repository/imagerepository picks the Config from repository/registry > kapp-ctrl deploys and knative runs the config > final pod running on the Kubernetes cluster.

Configure the developer workload in Tanzu Application Platform

The following YAML is an example `Workload` that adds the annotation `autoscaling.knative.dev/minScale = "1"` to set the minimum scale for the `spring-petclinic` app:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: spring-petclinic
  namespace: default
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  annotations:
    autoscaling.knative.dev/minScale: "1"
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/spring-petclinic
```

Deploy the workload

To deploy the workload, run:

```
kapp -y deploy -n default -a workloads -f workloads.yaml
```

Verify the annotation has propagated through the Supply Chain

To verify the annotation:

1. Verify that the workload build is successful by ensuring that `SUCCEEDED` is set to `True`:

   ```
   kubectl get builds
   NAME                          IMAGE
   SUCCEEDED
   spring-petclinic-build-1      dev.registry.tanzu.vmware.com/app-live-view/test/s
   pring-petclinic-default@sha256:9db2a8a8e77e9215239431fd8afe3f2ecdf09cce8afac565
   dad7b5f0c5ac0cdf     True
   ```

2. Verify the PodIntent of your workload by ensuring `status.template.metadata.annotations` shows the newly added annotation has propagated through the Supply Chain:

```
kubectl get podintents.conventions.carto.run spring-petclinic -oyaml

status:
  conditions:
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-12-03T15:14:33Z"
    status: "True"
    type: Ready
  observedGeneration: 3
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
```

3. Verify the ConfigMap was created for the app by ensuring `spec.template.metadata.annotations` shows the newly added annotation has propagated through the Supply Chain:

```
kubectl describe configmap spring-petclinic
Name:         spring-petclinic
Namespace:    default
Labels:       carto.run/cluster-supply-chain-name=source-to-url
              carto.run/cluster-template-name=config-template
              carto.run/resource-name=app-config
              carto.run/template-kind=ClusterConfigTemplate
              carto.run/workload-name=spring-petclinic
              carto.run/workload-namespace=default
Annotations:  <none>

Data
====
delivery.yml:
----
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: spring-petclinic
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
    kapp.k14s.io/app: "1638455805474051000"
    kapp.k14s.io/association: v1.5a9384bd7b93ca74ef494c4dec2caa4b
    tanzu.app.live.view: "false"
    app.kubernetes.io/component: run
    carto.run/workload-name: spring-petclinic
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/minScale: "1"
```

4. Verify the running Knative application pod by ensuring `annotations` shows the newly added annotation on the Knative application pod:

```
kubectl get pods -o yaml spring-petclinic-00002-deployment-77dbb85c65-cf7rn | g
rep annotations
  annotations:
    kapp.k14s.io/original: '{"apiVersion":"carto.run/v1alpha1","kind":"Workloa
d","metadata":{"annotations":{"autoscaling.knative.dev/minScale":"1"},"labels":
```

```
{"app.kubernetes.io/part-of":"tanzu-java-web-app","apps.tanzu.vmware.com/worklo
ad-type":"web","kapp.k14s.io/app":"1638455805474051000","kapp.k14s.io/associati
on":"v1.5a9384bd7b93ca74ef494c4dec2caa4b","tanzu.app.live.view":"false"},"nam
e":"spring-petclinic","namespace":"default"},"spec":{"source":{"git":{"ref":{"b
ranch":"main"},"url":"https://github.com/ksankaranara-vmw/spring-petclini
c"}}}}'
```

Your Knative app is now set to a minimum scale of one so that Application Live View can observe the instance of the app.

Application Live View on OpenShift

Application Live View must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards on Openshift. Tanzu Application Platform configures the following SCC for Application Live View back end, Application Live View connector, and Application Live View convention service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

The following is a `SecurityContextConstraints` specification for Application Live View connector:

```
---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: appliveview-connector-restricted-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
seccompProfiles:
  - runtime/default
```

The preceding `SecurityContextConstraints` specification is applicable to Application Live View back end and Application Live View convention service as well.

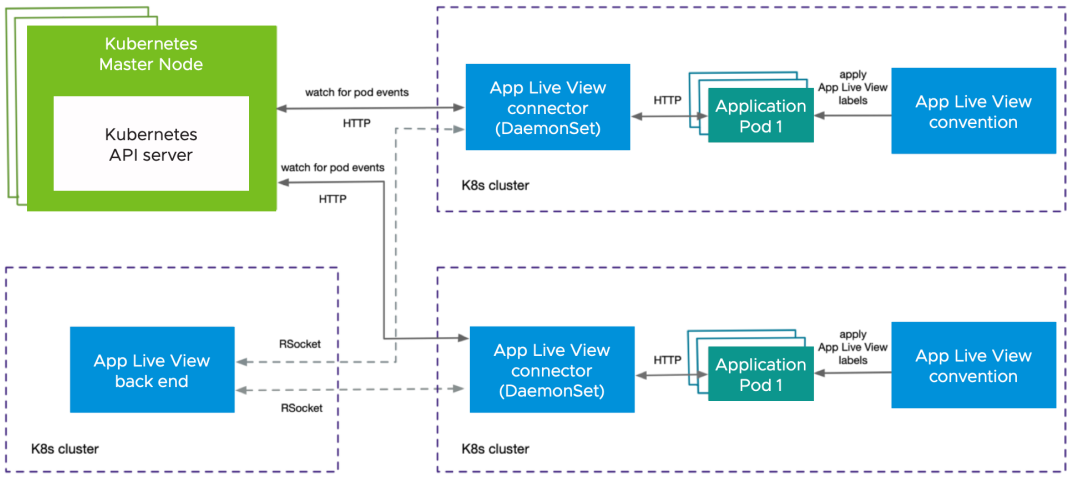
Support for polyglot apps with Application Live View

Application Live View currently supports Spring Boot, Spring Cloud Gateway, and Steeltoe apps.

- To enable Application Live View on Spring Boot and Spring Cloud Gateway apps, see Enable Application Live View for Spring Boot apps.
- To enable Application Live View on Steeltoe apps, see Enable Application Live View for Steeltoe apps.

Application Live View internal architecture

This topic describes the architecture of Application Live View and its components. You can deploy this system on a Kubernetes stack and use it to monitor containerized apps on hosted cloud platforms or on-premises.



Component overview

Application Live View includes the following components as shown in the architecture diagram:

- **Application Live View back end**

  Application Live View back end is the central server component that contains a list of registered apps. It provides a REST API that fetches the actuator data for the applications. The Application Live View UI plug-in, as part of Tanzu Application Platform GUI, queries this back-end REST API to get live actuator information for the pod.

- **Application Live View connector**

  Application Live View connector is the component responsible for discovering the app pods running on the Kubernetes cluster and registering the instances to the Application Live View back end for it to be observed. The Application Live View connector is also responsible for proxying the actuator queries to the app pods running in the Kubernetes cluster. The actuator data is then displayed in the Application Live View UI plug-in as part of Tanzu Application Platform GUI.

  You can deploy Application Live View connector in two modes:

  - `Cluster access`: Deploy as a Kubernetes DaemonSet to discover apps across all the namespaces running in a worker node of a Kubernetes cluster. This is the default mode of Application Live View connector.
  - `Namespace scoped`: Deploy as a Kubernetes Deployment to discover apps running within a namespace across worker nodes of Kubernetes cluster.

- **Application Live View convention server**

This component provides a webhook handler for the Tanzu convention controller. The webhook handler is registered with Tanzu convention controller. The webhook handler detects supply-chain workloads running a Spring Boot. Such workloads are annotated automatically to enable Application Live View to monitor them. Download and install the Application Live View conventions Webhook component with Tanzu Application Platform.

- **Application Live View APIServer**

  Application Live View APIServer generates a unique token when a user receives access validation to a pod. The Application Live View connector component verifies the token against the Application Live View APIServer before proxying the actuator data from the application. This ensures that the actuator data is secured and only the user who has valid access to view the live information for the pod can retrieve the data.

Design flow

As illustrated in the diagram, the applications run by the user are registered with Application Live View back end by using Application Live View connector. After the application is registered, the Application Live View back end offers the ability to serve actuator data from that registered application through its REST API. Application Live View back end proxies the call to the connector for querying actuator endpoint information.

Application Live View connector, which is a lean model, uses specific labels to discover apps across cluster or namespace. Application Live View connector serves as the connection between running applications and Application Live View back end. Application Live View connector communicates with the Kubernetes API server requesting events for pod creation and termination, and then filters out the events to find the pod of interest by using labels. Then Application Live View connector registers the filtered app instances with Application Live View back end.

Application Live View back end and Application Live View connector communicate through a bidirectional RSocket channel. Application Live View connector is implemented as a Java/Spring Boot application and runs as a native executable file (Spring Native using GraalVM). Application Live View connector runs as a DaemonSet by default on every node in the cluster.

Application Live View conventions identifies PodIntents for pods that can serve actuator data and annotates the PodSpec with application-specific labels. Those labels are used by the Application Live View connector to identify running pods that can serve actuator data. Application Live View conventions reads the image metadata to determine the application-specific labels applied on the PodSpec.

Troubleshoot Application Live View

This topic provides information to help you troubleshoot Application Live View.

App is not visible in Application Live View UI

Symptom

Your app is not visible in the Application Live View UI.

Solution

The connector component is responsible for discovering the app and registering it with Application Live View.

To troubleshoot, confirm the following:

1. The app must be a Spring Boot Application.

2. Confirm that an instance of a connector is located in the same namespace as your app.

   ```
   kubectl get pods -n NAMESPACE | grep connector
   ```

   Where `NAMESPACE` is the name of the namespace that your app is located in.

3. Confirm that the actuator endpoints are enabled for your app as follows:

   ```
   management.endpoints.web.exposure.include: "*"
   ```

4. Confirm that you have included the following labels within your app deployment YAML file:

   ```
   tanzu.app.live.view="true"
   tanzu.app.live.view.application.name="APP-NAME"
   ```

   Where `APP-NAME` is the name of your app.

5. Confirm that the Convention Service workload YAML file does not contain property `management.endpoints.web.exposure.include` overrides.

See also:

- App is not visible in Application Live View UI with actuator endpoints enabled
- The UI does not show any information for an app with actuator endpoints exposed at root

App is not visible in Application Live View UI with actuator endpoints enabled

Symptom

Your app is not visible in Application Live View UI, but the actuator endpoints are enabled.

Solution

To troubleshoot:

1. Check the port on which actuator endpoints are configured. By default, they are enabled on the application port. If they are configured on a port different from the application port, set the labels in your app deployment YAML file as follows:

   ```
   tanzu.app.live.view.application.port: "APPLICATION-PORT"
   tanzu.app.live.view.application.actuator.port: "ACTUATOR-PORT"
   ```

   Where:

   - `APPLICATION-PORT` is the application port.
   - `ACTUATOR-PORT` is the actuator port.

2. Check the path on which the app and actuator endpoints are configured. If they are configured on a different paths, set the labels in your app deployment YAML file as follows:

   ```
   tanzu.app.live.view.application.path: "APPLICATION-PATH"
   tanzu.app.live.view.application.actuator.path: "ACTUATOR-PATH"
   ```

   Where:

   - `APPLICATION-PATH` is the application port.
   - `ACTUATOR-PATH` is the actuator port.

The UI does not show any information for an app with actuator endpoints exposed at root

Symptom

Your app has actuator endpoints exposed at root and the UI does not show any information.

Cause

Application Live View cannot display the app details when the app is exposing the actuator endpoint on root (`/`) . This is due to conflict in the actuator context path and app default context path.

No information shown on the Health page

Symptom

The app shows up in Application Live View UI, but the **Health** page does not show details of health.

Solution

The information exposed by the health endpoint depends on the `management.endpoint.health.show-details` property. This must be set to `always` as as follows:

```
management.endpoint.health.show-details: "always"
```

Stale information in Application Live View

Symptom

You can find your app in the UI, but it is an old instance that no longer exists while the new instance doesn't show up yet.

Solution

To troubleshoot:

1. View the Application Live View connector pod logs to see if the connector is sending updates to the back end.
2. Delete the connector pod to recreate it by running:

   ```
   kubectl -n app-live-view-connector delete pods -l=name=application-live-view-co
   nnector
   ```

Unable to find CertificateRequests in Application Live View convention

Symptom

The certificate request is missing for certificate `app-live-view-conventions/appliveview-webhook-cert`.

Solution

To troubleshoot:

1. Run `kubectl get certificaterequest -A` to see if the certificate request is missing for Application Live View convention.

2. Delete the secret `appliveview-webhook-cert` that corresponds to the certificate in the `app-live-view-conventions` namespace by running:

   ```
   kubectl delete secret appliveview-webhook-cert -n app-live-view-conventions
   ```

   This recreates the certificate request and updates the corresponding certificate.

No live information for pod with ID

Symptom

In Tanzu Application Platform GUI, you receive the error `No live information for pod with id`.

Cause

This might happen because of stale information in Application Live View because it is an old instance that no longer exists while the new instance doesn't show up yet.

Solution

The workaround is to delete the connector pod so it is re-created by running:

```
kubectl -n app-live-view-connector delete pods -l=name=application-live-view-connector
```

Cannot override the actuator path in the labels

Symptom

You are unable to override the actuator path in the labels as part of the workload deployment.

Cause

The changes to add or override the labels or annotations in the `Workload` are in progress. The changes from the `Workload` must be propagated up through the supply chain for the PodIntent to see the new changes.

Cannot configure SSL in appliveview-connector

Symptom

This might be because `sslDeactivated` flag in the values YAML file does not accept values without quotes.

Cause

The `sslDeactivated` Boolean flag is treated as a string in the Kubernetes YAML file.

Solution

You must specify the value within double quotation marks for the configuration to be picked up.

Verify the labels in your workload YAML file

To verify that the labels in your workload YAML file are working:

1. Verify the app live view convention webhook is running properly by running:

   ```
   kubectl get pods -n app-live-view | grep webhook
   ```

2. Verify the conventions controller is running properly by running:

```
kubectl get pods -n conventions-system
```

3. Verify that the conventions are applied properly to the PodSpec by running:

   ```
   kubectl get podintents.conventions.carto.run WORKLOAD-NAME -oyaml
   ```

   Where `WORKLOAD-NAME` is the name of your workload.

   If everything works correctly, the status will contain a transformed template that includes the labels added as part of your workload YAML file. For example:

   ```
   status:
   conditions:
   - lastTransitionTime: "2021-10-26T11:26:35Z"
     status: "True"
     type: ConventionsApplied
   - lastTransitionTime: "2021-10-26T11:26:35Z"
     status: "True"
     type: Ready
   observedGeneration: 1
   template:
     metadata:
       annotations:
         conventions.carto.run/applied-conventions: |-
           appliveview-sample/app-live-view-connector
           appliveview-sample/app-live-view-appflavours
           appliveview-sample/app-live-view-systemproperties
       labels:
         tanzu.app.live.view: "true"
         tanzu.app.live.view.application.flavours: spring-boot
         tanzu.app.live.view.application.name: petclinic
     spec:
       containers:
       - env:
         - name: JAVA_TOOL_OPTIONS
           value: -Dmanagement.endpoint.health.show-details=always -Dmanagement.en
   dpoints.web.exposure.include=*
       image: index.docker.io/kdvolder/alv-spring-petclinic:latest@sha256:1aa7bd22
   8137471ea38ce36cbf5ffcd629eabeb8ce047f5533b7b9176ff51f98
       name: workload
       resources: {}
   ```

Override labels set by the Application Live View convention service

It is not possible to override the labels set by the Application Live View convention service for the workload deployment in Tanzu Application Platform. The labels `tanzu.app.live.view`, `tanzu.app.live.view.application.flavours` and `tanzu.app.live.view.application.name` cannot be overridden. The default values set by the Application Live View convention server are used.

However, if you want to override `management.endpoints.web.exposure.include` or `management.endpoint.health.show-details`, you can override these environment properties in `application.properties` or `application.yml` in the Spring Boot Application before deploying the workload in Tanzu Application Platform. Environment properties updated in your app take precedence over the default values set by Application Live View convention server.

Configure labels when management.endpoints.web.base-path and management.server.port are set

If the custom actuator context path is configured as follows:

```
management.endpoints.web.base-path=/manage
management.server.port=8085
```

Configure the connector as follows:

```
tanzu.app.live.view.application.actuator.path=/manage   (manage is the custom actuator
path set on the application)
tanzu.app.live.view.application.actuator.port=8085   (8085 is the custom management se
rver port set on the application)
```

Uninstall Application Live View

This topic tells you how to uninstall Application Live View from Tanzu Application Platform (commonly known as TAP).

To uninstall the Application Live View back end, Application Live View connector, and Application Live View convention server, run:

```
tanzu package installed delete appliveview -n tap-install
tanzu package installed delete appliveview-connector -n tap-install
tanzu package installed delete appliveview-conventions -n tap-install
```

Overview of Application Single Sign-On for VMware Tanzu® 3.1

Application Single Sign-On for VMware Tanzu® (AppSSO) provides APIs for curating and consuming a “Single Sign-On as a service” offering on Tanzu Application Platform.

To get started with AppSSO, see Get started with Application Single Sign-On.

With AppSSO, Service Operators can configure and deploy authorization servers. Application Operators can then configure their Workloads with these authorization servers to provide Single Sign-On to their end-users.

AppSSO allows integrating authentication and authorization decisions early in the software development and release life cycle. It provides a seamless transition for workloads from development to production when including Single Sign-On solutions in your software.

It’s easy to get started with AppSSO, deploy an authorization server with static test users, and eventually progress to multiple authorization servers of production-grade scale with token key rotation, multiple upstream identity providers, configured secure storage, and client restrictions.

AppSSO’s authorization server is based off of Spring Authorization Server project. For more information, see Spring documentation.

Overview of Application Single Sign-On for VMware Tanzu® 3.1

Application Single Sign-On for VMware Tanzu® (AppSSO) provides APIs for curating and consuming a “Single Sign-On as a service” offering on Tanzu Application Platform.

To get started with AppSSO, see Get started with Application Single Sign-On.

With AppSSO, Service Operators can configure and deploy authorization servers. Application Operators can then configure their Workloads with these authorization servers to provide Single Sign-On to their end-users.

AppSSO allows integrating authentication and authorization decisions early in the software development and release life cycle. It provides a seamless transition for workloads from development to production when including Single Sign-On solutions in your software.

It's easy to get started with AppSSO, deploy an authorization server with static test users, and eventually progress to multiple authorization servers of production-grade scale with token key rotation, multiple upstream identity providers, configured secure storage, and client restrictions.

AppSSO's authorization server is based off of Spring Authorization Server project. For more information, see Spring documentation.

Get started with Application Single Sign-On

This topic tells you about concepts important to getting started with Application Single Sign-On (commonly called AppSSO).

Use this topic to learn how to:

1. Set up your first authorization server.
2. Provision a ClientRegistration.
3. Deploy an application that uses the provisioned ClientRegistration to enable SSO.

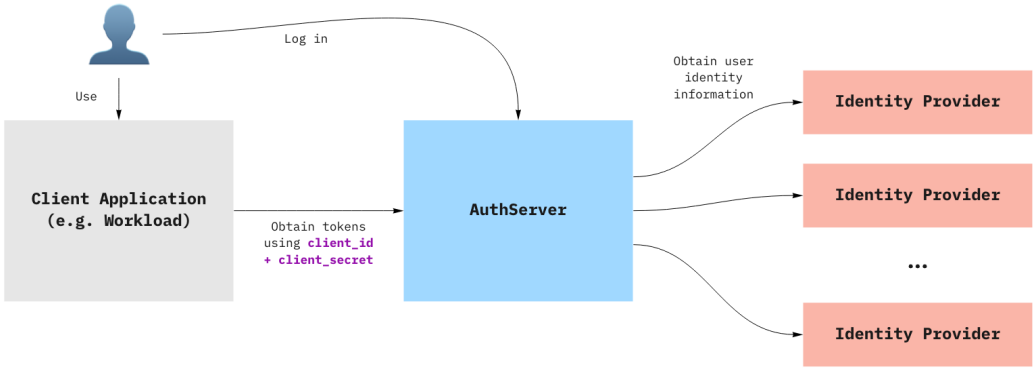
After completing these steps, you can proceed with securing a Workload.

Prerequisites

You must install AppSSO on your Tanzu Application Platform cluster. For more information, see Install AppSSO.

Key concepts

At the core of AppSSO is the concept of an Authorization Server, outlined by the AuthServer custom resource. Service Operators create those resources to provision running Authorization Servers, which are OpenID Connect Providers. They issue ID Tokens to Client applications, which contain identity information about the end user such as email, first name, last name and so on.



When a Client application uses an AuthServer to authenticate an End-User, the typical steps are:

1. The End-User visits the Client application
2. The Client application redirects the End-User to the AuthServer, with an OAuth2 request
3. The End-User logs in with the AuthServer, usually using an external Identity Provider (e.g. Google, Azure AD)
    1. Identity Providers are set up by Service Operators

    2. AuthServers may use various protocols to obtain identity information about the user, such as OpenID Connect, SAML or LDAP, which may involve additional redirects

4. The AuthServer redirects the End-User to the Client application with an authorization code
5. The Client application exchanges with the AuthServer for an `id_token`
    1. The Client application does not know how the identity information was obtained by the AuthServer, it only gets identity information in the form of an ID Token.

ID Tokens are JSON Web Tokens containing standard Claims about the identity of the user (e.g. name, email, etc) and about the token itself (e.g. “expires at”, “audience”, etc.). Here is an example of an `id_token` as issued by an Authorization Server:

```
{
        "iss": "https://appsso.example.com",
        "sub": "213435498y",
        "aud": "my-client",
        "nonce": "fkg0-90_mg",
        "exp": 1656929172,
        "iat": 1656928872,
        "name": "Jane Doe",
        "given_name": "Jane",
        "family_name": "Doe",
        "email": "jane.doe@example.com",
        "roles": [
                "developer",
                "org-user"
        ]
}
```

`roles` claim can only be part of an `id_token` when user roles are mapped and ‘roles’ scope is requested. For more information about mapping for OpenID Connect, LDAP and SAML, see:

- OpenID external groups mapping
- LDAP external groups mapping
- SAML (experimental) external groups mapping

ID Tokens are signed by the `AuthServer`, using Token Signature Keys. Client applications may verify their validity using the AuthServer’s public keys.

Next steps

- Provision an AuthServer

Get started with Application Single Sign-On

This topic tells you about concepts important to getting started with Application Single Sign-On (commonly called AppSSO).

Use this topic to learn how to:

1. Set up your first authorization server.
2. Provision a ClientRegistration.
3. Deploy an application that uses the provisioned ClientRegistration to enable SSO.

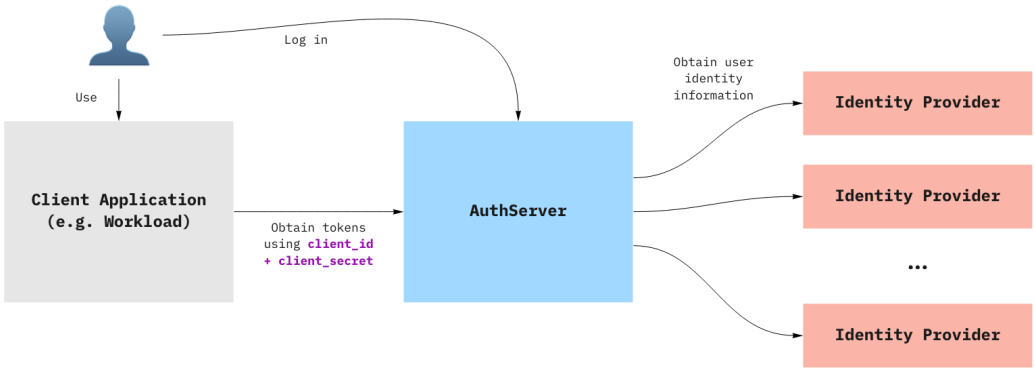
After completing these steps, you can proceed with securing a Workload.

Prerequisites

You must install AppSSO on your Tanzu Application Platform cluster. For more information, see Install AppSSO.

Key concepts

At the core of AppSSO is the concept of an Authorization Server, outlined by the AuthServer custom resource. Service Operators create those resources to provision running Authorization Servers, which are OpenID Connect Providers. They issue ID Tokens to Client applications, which contain identity information about the end user such as email, first name, last name and so on.



When a Client application uses an AuthServer to authenticate an End-User, the typical steps are:

1. The End-User visits the Client application
2. The Client application redirects the End-User to the AuthServer, with an OAuth2 request
3. The End-User logs in with the AuthServer, usually using an external Identity Provider (e.g. Google, Azure AD)
    1. Identity Providers are set up by Service Operators
    2. AuthServers may use various protocols to obtain identity information about the user, such as OpenID Connect, SAML or LDAP, which may involve additional redirects
4. The AuthServer redirects the End-User to the Client application with an authorization code
5. The Client application exchanges with the AuthServer for an `id_token`
    1. The Client application does not know how the identity information was obtained by the AuthServer, it only gets identity information in the form of an ID Token.

ID Tokens are JSON Web Tokens containing standard Claims about the identity of the user (e.g. name, email, etc) and about the token itself (e.g. “expires at”, “audience”, etc.). Here is an example of an `id_token` as issued by an Authorization Server:

```
{
        "iss": "https://appsso.example.com",
        "sub": "213435498y",
        "aud": "my-client",
        "nonce": "fkg0-90_mg",
        "exp": 1656929172,
        "iat": 1656928872,
        "name": "Jane Doe",
        "given_name": "Jane",
        "family_name": "Doe",
        "email": "jane.doe@example.com",
        "roles": [
                "developer",
                "org-user"
```

```
        ]
}
```

`roles` claim can only be part of an `id_token` when user roles are mapped and 'roles' scope is requested. For more information about mapping for OpenID Connect, LDAP and SAML, see:

- OpenID external groups mapping
- LDAP external groups mapping
- SAML (experimental) external groups mapping

ID Tokens are signed by the `AuthServer`, using Token Signature Keys. Client applications may verify their validity using the AuthServer's public keys.

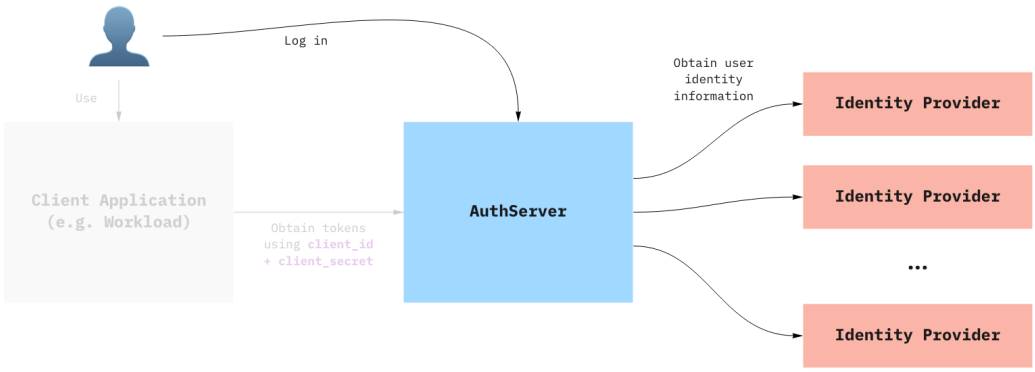
Next steps

- Provision an AuthServer

Provision an AuthServer

This topic tells you how to provision an AuthServer for Application Single Sign-On (commonly called AppSSO). Use this topic to learn how to:

1. Set up your first authorization server in the `default` namespace.
2. Ensure it is running so that users can log in.



Prerequisites

You must install AppSSO on your Tanzu Application Platform cluster and ensure that your Tanzu Application Platform installation is correctly configured.

AppSSO is installed with the `run`, `iterate`, and `full` profiles, no extra steps required.

To verify AppSSO is installed on your cluster, run:

```
tanzu package installed list -A | grep "sso.apps.tanzu.vmware.com"
```

For more information about the Tanzu Application Platform installation, see Install Tanzu Application Platform.

For more information about the AppSSO installation, see Install AppSSO.

Provision an AuthServer

Deploy your first Authorization Server along with an `RSAKey` key for signing tokens.

Caution

This `AuthServer` example uses an unsafe testing-only identity provider. Never use it in production environments. For more information about identity providers, see Identity providers.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: my-authserver-example
  namespace: default
  labels:
    name: my-first-auth-server
    env: tutorial
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "default"
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  replicas: 1
  tls:
    deactivated: true
  identityProviders:
    - name: "internal"
      internalUnsafe:
        users:
          - username: "user"
            password: "password"
            email: "user@example.com"
            emailVerified: true
            roles:
              - "user"
  tokenSignature:
    signAndVerifyKeyRef:
      name: "authserver-signing-key"

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: authserver-signing-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

You can wait for the `AuthServer` to become ready with:

```
kubectl wait --for=condition=Ready authserver my-authserver-example
```

Alternatively, you can inspect your `AuthServer` like any other resource:

```
kubectl get authservers.sso.apps.tanzu.vmware.com --all-namespaces
```

and you should see:

```
NAMESPACE NAME                  REPLICAS ISSUER URI
CLIENTS STATUS
default   my-authserver-example 1        http://my-authserver-example.default.<your do
main> 0       Ready
```

As you can see your `AuthServer` gets an issuer URI templated. This is its entrypoint. You can find an `AuthServer`'s issuer URI in its status:

```
kubectl get authservers.sso.apps.tanzu.vmware.com my-authserver-example -o jsonpath
='{.status.issuerURI}'
```

Open your `AuthServer`'s issuer URI in your browser. You should see a login page. Log in using username = `user` and password = `password`.

You can review the standard OpenID information of your `AuthServer` by visiting `http://my-authserver-example.default.<your domain>/.well-known/openid-configuration`.

Important

If the issuer URIs domain is not yours, the AppSSO package installation must be updated. For more information, see Install Application Single Sign-On.

The AuthServer spec in detail

Here is a detailed explanation of the `AuthServer` you have applied in the above section. This is intended to give you an overview of the different configuration values that were passed in. It is not intended to describe all the ins-and-outs, but there are links to related docs in each section.

Feel free to skip ahead.

Metadata

```
metadata:
  labels:
    name: my-first-auth-server
    env: tutorial
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "default"
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
```

The `metadata.labels` uniquely identify the AuthServer. They are used as selectors by `ClientRegistrations`, to declare from which authorization server a specific client obtains tokens from.

The `sso.apps.tanzu.vmware.com/allow-client-namespaces` annotation restricts the namespaces in which you can create a `ClientRegistrations` targeting this authorization server. In this case, the authorization server will only pick up client registrations in the `default` namespace.

The `sso.apps.tanzu.vmware.com/allow-unsafe-...` annotations enable "development mode" features, useful for testing. Those should not be used for production-grade authorization servers.

For more information about annotations and labels in `AuthServer` resource, see Annotation and labels.

TLS & issuer URI

```
spec:
  tls:
    deactivated: true
```

The `tls` field configures whether and how to obtain a certificate for an `AuthServer` to secure its issuer URI. If you deactivate `tls`, the issuer URI uses plain HTTP.

Caution

Plain HTTP access is for development purposes only and must never be used in production. For more information about the production readiness with TLS, see Issuer URI & TLS.

Token Signature

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
# ...
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: "authserver-signing-key"
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: authserver-signing-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

The token signing key is the private RSA key used to sign ID Tokens, using JSON Web Signatures, and clients use the public key to verify the provenance and integrity of the ID tokens. The public keys used for validating messages are published as JSON Web Keys at `{.status.issuerURI}/oauth2/jwks`.

The `spec.tokenSignature.signAndVerifyKeyRef.name` references a secret containing PEM-encoded RSA keys, both `key.pem` and `pub.pem`. In this specific example, we are using Secretgen-Controller, a TAP dependency, to generate the key for us.

Learn more about Token Signatures.

Identity providers

```
spec:
  identityProviders:
    - name: "internal"
      internalUnsafe:
        users:
          - username: "user"
            password: "password"
            email: "user@example.com"
            roles:
              - "user"
```

AppSSO's authorization server delegates login and user management to external identity providers (IDP), such as Google, Azure Active Directory, Okta and so on. See diagram at the top of this topic for more information.

In this example, we use an `internalUnsafe` identity provider. As the name implies, it is *not* an external IDP, but rather a list of hardcoded user/passwords. As the name also implies, this is not considered safe for production. Here, we declared a user with username = `user`, and password = `password`. For production setups, consider using OpenID Connect IDPs instead.

The `email` and `roles` fields are optional for internal users. However, they will be useful when we want to use SSO with a client application later in this guide.

Caution

VMware discourages using the `internalUnsafe` identity provider in production environments.

Configuring storage

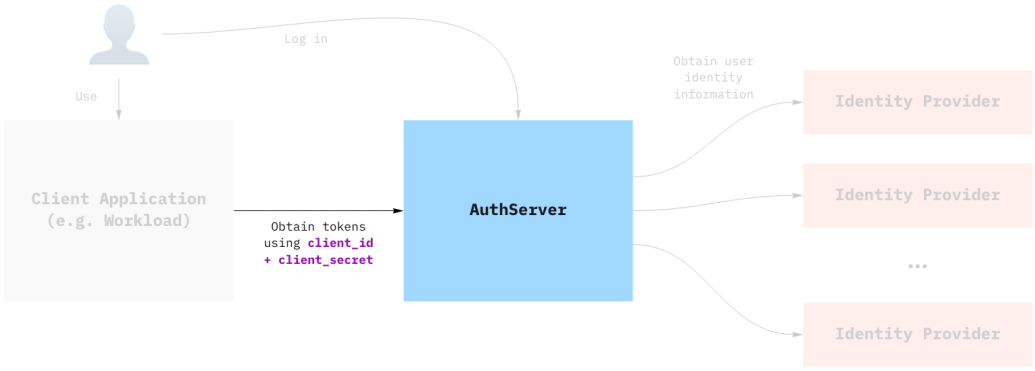
An `AuthServer` issues a Redis instance by default. It can be used for testing, prototyping and other non-production purposes. No additional configuration is required.

To configure your own storage that is ready for production, see Storage.

Provision a client registration

This topic tells you how to provision a client registration for Application Single Sign-On (commonly called AppSSO). Use this topic to learn how to:

1. Obtain credentials for the Authorization Server that you provisioned in Provision your first AuthServer.
2. Verify that the credentials are valid using client-credentials flow.



Prerequisites

Complete the steps described in Get started with Application Single Sign-On.

Creating the ClientRegistration

Assuming you have deployed the AuthServer as described previously, you can create and apply the following client registration:

Note

AppSSO uses `test-app.example.com` for `ClientRegistration.spec.redirectURIs[0]`. You must customize the URL to match the domain of your Tanzu Application Platform cluster. This is the URL to expose your test application in the next section.

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
   name: my-client-registration
   namespace: default
spec:
   authServerSelector:
      matchLabels:
         name: my-first-auth-server
         env: tutorial
   redirectURIs:
      - "http://test-app.example.com/oauth2/callback"
   requireUserConsent: false
   clientAuthenticationMethod: client_secret_basic
   authorizationGrantTypes:
      - "client_credentials"
      - "authorization_code"
   scopes:
      - name: "openid"
      - name: "email"
      - name: "profile"
      - name: "roles"
      - name: "message.read"
```

The AuthServer should now have this `ClientRegistration` registered. You can verify the status either by looking at the `ClientRegistration`s `.status` field, or looking at the `AuthServer` itself.

```
# Check the client registration
kubectl get clientregistration my-client-registration -n default -o yaml
# Check the authserver
kubectl get authservers
# NAME                    REPLICAS   ISSUER URI                      CLIENTS   TOKEN KE
YS
# my-authserver-example   1          http://authserver.example.com  1         1
#                                                                   ^
#                                 the AuthServer now has one client ^
```

AppSSO will create a secret containing the credentials that client applications will use, named after the client registration. The type of the secret is `servicebinding.io/oauth2`. You can obtain the values in the secret by running:

```
kubectl get secret my-client-registration -n default  -o json | jq ".data | map_values
(@base64d)"
# {
#   "authorization-grant-types": "client_credentials,authorization_code",
#   "client-authentication-method": "client_secret_basic",
#   "client-id": "default_my-client-registration",
#   "client-secret": "PLACEHOLDER",
#   "issuer-uri": "http://authserver.example.com",
#   "provider": "appsso",
#   "scope": "openid,email,profile,roles,message.read",
#   "type": "oauth2"
# }
```

Validating that the credentials are working

Before you deploy an app and make use of SSO, you can try the credentials from your machine to try and obtain an `access_token` using the `client_credentials` grant. You need the client_id and client_secret that were created as part of the client registration.

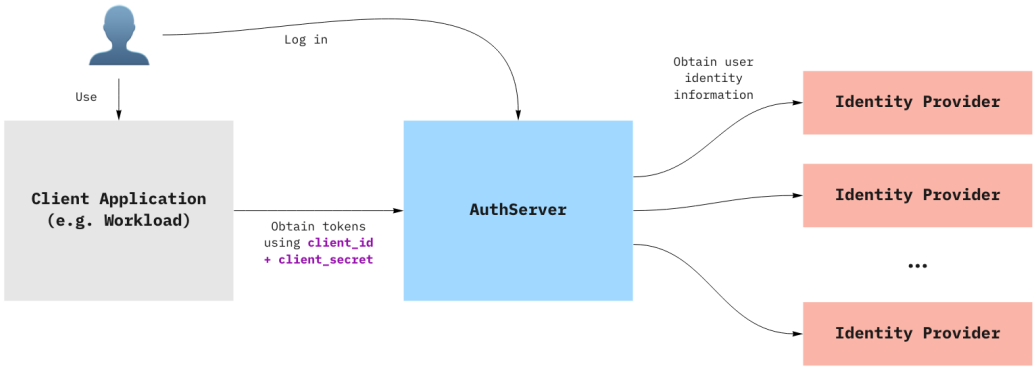
```
CLIENT_ID=$(kubectl get secret my-client-registration -n default -o jsonpath="{.data.client-id}" | base64 -d)
CLIENT_SECRET=$(kubectl get secret my-client-registration -n default -o jsonpath="{.data.client-secret}" | base64 -d)
ISSUER_URI=$(kubectl get secret my-client-registration -n default -o jsonpath="{.data.issuer-uri}" | base64 -d)
curl -XPOST "$ISSUER_URI/oauth2/token?grant_type=client_credentials&scope=message.read" -u "$CLIENT_ID:$CLIENT_SECRET"
```

You can decode the `access_token` using an online service, such as JWT.io.

To learn more about grant types, see Grant Types

Deploy an application with Application Single Sign-On

This topic tells you how to deploy a minimal Kubernetes application that is protected by Application Single Sign-On (commonly called AppSSO) by using the credentials that ClientRegistration creates.



For more information about how a Client application uses an AuthServer to authenticate an end user, see AppSSO Overview.

Prerequisites

You must complete the steps described in Get started with Application Single Sign-On. If not, see Provision a client registration.

Deploy a minimal application

You are going to deploy a two-container pod, as a test application.

Important

AppSSO uses `test-app.example.com` for `HTTPProxy.spec.virtualhost.fqdn`. You must customize the URL to match the domain of your Tanzu Application Platform cluster. This URL must match what was set up in `ClientRegistration.spec.redirectURIs[0]` in Provision a client registration

```
---
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: test-application
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      name: test-application
  template:
    metadata:
      labels:
        name: test-application
    spec:
      containers:
        - image: bitnami/oauth2-proxy:7.3.0
          name: proxy
          ports:
            - containerPort: 4180
              name: proxy-port
              protocol: TCP
          env:
            - name: ISSUER_URI
              valueFrom:
                secretKeyRef:
                  name: my-client-registration
                  key: issuer-uri
            - name: CLIENT_ID
              valueFrom:
                secretKeyRef:
                  name: my-client-registration
                  key: client-id
            - name: CLIENT_SECRET
              valueFrom:
                secretKeyRef:
                  name: my-client-registration
                  key: client-secret
          command: [ "oauth2-proxy" ]
          args:
            - --oidc-issuer-url=$(ISSUER_URI)
            - --client-id=$(CLIENT_ID)
            - --insecure-oidc-skip-issuer-verification=true
            - --client-secret=$(CLIENT_SECRET)
            - --cookie-secret=0000000000000000
            - --cookie-secure=false
            - --http-address=http://:4180
            - --provider=oidc
            - --scope=openid email profile roles
            - --email-domain=*
            - --insecure-oidc-allow-unverified-email=true
            - --oidc-groups-claim=roles
            - --upstream=http://127.0.0.1:8000
            - --redirect-url=http://test-app.example.com/oauth2/callback
            - --skip-provider-button=true
            - --pass-authorization-header=true
            - --prefer-email-to-user=true
        - image: python:3.9
          name: application
          resources:
            limits:
              cpu: 100m
              memory: 100Mi
          command: [ "python" ]
          args:
            - -c
            - |
              from http.server import HTTPServer, BaseHTTPRequestHandler
```

```
              import base64
              import json

              class Handler(BaseHTTPRequestHandler):
                  def do_GET(self):
                      if self.path == "/token":
                          self.token()
                          return
                      else:
                          self.greet()
                          return

                  def greet(self):
                      username = self.headers.get("x-forwarded-user")
                      self.send_response(200)
                      self.send_header("Content-type", "text/html")
                      self.end_headers()
                      page = f"""
                      <h1>It Works!</h1>
                      <p>You are logged in as <b>{username}</b></p>
                      """
                      self.wfile.write(page.encode("utf-8"))

                  def token(self):
                      token = self.headers.get("Authorization").split("Bearer ")[-1]
                      payload = token.split(".")[1]
                      decoded = base64.b64decode(bytes(payload, "utf-8") + b'==').deco
de("utf-8")
                      self.send_response(200)
                      self.send_header("Content-type", "application/json")
                      self.end_headers()
                      self.wfile.write(decoded.encode("utf-8"))

              server_address = ('', 8000)
              httpd = HTTPServer(server_address, Handler)
              httpd.serve_forever()

---
apiVersion: v1
kind: Service
metadata:
  name: test-application
  namespace: default
spec:
  ports:
    - port: 80
      targetPort: 4180
  selector:
    name: test-application

---
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: test-application
  namespace: default
spec:
  virtualhost:
    fqdn: test-app.example.com
  routes:
    - conditions:
        - prefix: /
      services:
        - name: test-application
          port: 80
```

Now you can navigate to `http://test-app.example.com/`. It may ask you to log into the AuthServer you haven't already. You can also navigate to `http://test-app.example.com/token` if you wish to see the contents of the ID token.

Deployment manifest

The application was deployed as a two-container pod: one for the app, and one for handling login.

- The main container is called `application`, and runs a bare-bones Python HTTP server, that reads from the `Authorization` header from incoming requests and returns the decoded `id_token`.
- The second container, called `proxy`, is a sidecar container, an "Ambassador". It receives traffic for the Pod, performs OpenID authentication using OAuth2 Proxy, and proxies requests to the `application` with some added headers containing identity information.

Along with this deployment, there is a `Service` + `HTTPProxy`, to expose the application to the outside world.

OAuth2-Proxy

The setup of the above OAuth2 Proxy is minimal, and is not considered suitable for production use. To configure it for production, please refer to the official documentation.

Note that OAuth2 Proxy requires some claims to be present in the `id_token`, notably the `email` claim and the non-standard `groups` claim. The `groups` claim maps to AppSSO's `roles` claim. Therefore, for this proxy to work with AppSSO, users *MUST* have an e-mail defined, and at least one entry in `roles`. If the proxy container logs an error stating `Error redeeming code during OAuth2 callback: could not get claim "groups" [...]`, make sure that the user has `roles` provided in the `identityProvider`.

Application Single Sign-On for Platform Operators

This topic tells you how to manage the Application Single Sign-On (commonly called AppSSO) package installation and what it installs. Use this topic to learn:

- Install Application Single Sign-On
- Configure Application Single Sign-On
- RBAC for Application Single Sign-On
- Application Single Sign-On for OpenShift clusters
- Upgrade Application Single Sign-On
- Uninstall Application Single Sign-On

Application Single Sign-On for Platform Operators

This topic tells you how to manage the Application Single Sign-On (commonly called AppSSO) package installation and what it installs. Use this topic to learn:

- Install Application Single Sign-On
- Configure Application Single Sign-On
- RBAC for Application Single Sign-On
- Application Single Sign-On for OpenShift clusters
- Upgrade Application Single Sign-On

- Uninstall Application Single Sign-On

Install Application Single Sign-On

This topic tells you how to install Application Single Sign-On (commonly called AppSSO) from the Tanzu Application Platform (commonly called TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Application Single Sign-On. For more information about profiles, see Components and installation profiles.

What's inside

The AppSSO package will install the following resources:

- The `appsso` Namespace with a Deployment of the AppSSO controller and Services for Webhooks
- A `ServiceAccount` with RBAC outlined in detail here
- AuthServer and ClientRegistration CRDs

Prerequisites

Before installing AppSSO, please ensure you have Tanzu Application Platform installed on your Kubernetes cluster.

Installation

1. Learn more about the AppSSO package:

   ```
   tanzu package available get sso.apps.tanzu.vmware.com --namespace tap-install
   ```

2. Install the AppSSO package:

   ```
   tanzu package install appsso \
    --namespace tap-install \
    --package sso.apps.tanzu.vmware.com \
    --version 3.1
   ```

3. Confirm the package has reconciled successfully:

   ```
   tanzu package installed get appsso --namespace tap-install
   ```

Configure Application Single Sign-On

This topic tells you how to configure Application Single Sign-On (commonly called AppSSO).

TAP values

Most commonly, the AppSSO package installation is configured through TAP's meta package installation. The TAP package has a `shared` top-level configuration key for sharing common configuration between the packages it installs.

AppSSO inherits the `shared.{ingress_domain, ingress_issuer, ca_cert_data, kubernetes_distribution}` configuration values from Tanzu Application Platform. You can configure the AppSSO-specific parameters under `appsso`. AppSSO-specific configuration has precedence over the shared values of Tanzu Application Platform.

For example:

```
#! my-tap-values.yaml

shared:
# Shared configuration goes here.

appsso:
# AppSSO-specific configuration goes here.
```

domain_name

The AppSSO package has one required configuration value, its `domain_name`. It is used for templating the issuer URI for `AuthServer`. `domain_name` must be the shared ingress domain of your TAP package installation. If your TAP installation is configured with `shared.ingress_domain`, then AppSSO will inherit the correct configuration.

Note

If omitted, `domain_name` is set to `shared.ingress_domain`.

domain_template

You can customize how AppSSO template's issuerURIs with the `domain_template` configuration. This is a Golang text/template. The default is `"{{.Name}}.{{.Namespace}}.{{.Domain}}"`.

The domain template will be applied with the given `domain_name` and the `AuthServer`'s name and namespace:

- `{{.Domain}}` will evaluate to the configured `domain_name`
- `{{.Name}}` will evaluate to `AuthServer.metadata.name`
- `{{.Namespace}}` will evaluate to `AuthServer.metadata.namespace`

To be able to use a wild-card certificate, consider `"{{.Name}}-{{.Namespace}}.{{.Domain}}"`.

It is strongly recommended to keep the name and namespace part of the template to avoid domain name collisions.

default_authserver_clusterissuer

You can denote a `cert-manager.io/v1/ClusterIssuer` as a default issuer for `AuthServer.spec.tls.issuerRef` and omit `AuthServer.spec.tls`. When the value of `AuthServer.spec.tls.issuerRef` is the empty string `""`, no default issuer is assumed and `AuthServer.spec.tls` is required.

If you configured `shared.ingress_issuer` and omitted `default_authserver_clusterissuer` while installing Tanzu Application Platform, AppSSO uses the ingress issuer of Tanzu Application Platform and sets `default_authserver_clusterissuer` to `shared.ingress_issuer`.

ca_cert_data

You can configure trust for custom CAs by providing their certificates as a PEM bundle to `ca_cert_data`. As a result, all `AuthServer`s trust your custom CAs.

This is useful if you have identity providers serving certificates from a custom CA and configuring AuthServer storage.

Alternatively, you can configure trust for a single AuthServer.

Note

AppSSO-specific `ca_cert_data` is concatenated with `shared.ca_cert_data`. The resulting PEM bundle contains both.

kubernetes_distribution

This setting toggles behavior specific to Kubernetes distribution. Currently, the only supported values are `""` and `openshift`.

AppSSO installs *OpenShift*-specific RBAC and resources.

Note

If omitted, `kubernetes_distribution` is set to `shared.kubernetes_distribution`.

Configuration schema

The entire available configuration schema for AppSSO is:

```
#@schema/desc "Optional: Kubernetes platform distribution that this package is being i
nstalled on. Accepted values: ['','openshift']"
kubernetes_distribution: ""

#@schema/desc "Domain name for AuthServers"
domain_name: "example.com"

#@schema/desc "Optional: Golang template/text string for constructing AuthServer FQDN
s"
domain_template: "{{.Name}}.{{.Namespace}}.{{.Domain}}"

#@schema/desc "Optional: PEM-encoded certificate data for AuthServers to trust TLS con
nections with a custom CA"
ca_cert_data: ""

#@schema/desc "Optional: Interval at which the controller will re-synchronize applied
resources"
resync_period: "2h"

#@schema/desc "Optional: Number of controller replicas to deploy"
replicas: 1

#@schema/desc "Optional: Resource requirements the controller deployment"
resources:
  requests:
    #@schema/desc "CPU request of the controller"
    cpu: "20m"
    #@schema/desc "Memory request of the controller"
    memory: "100Mi"
  limits:
    #@schema/desc "CPU limit of the controller"
```

```
    cpu: "500m"
    #@schema/desc "Memory limit of the controller"
    memory: "500Mi"

#@schema/desc "Optional: Schema-free extension point for internal, package-private con
figuration (Unsupported! Use at your own risk.)"
#@schema/type any=True
internal: { }
```

RBAC for Application Single Sign-On

The AppSSO package aggregates the following permissions into TAP's well-known roles:

- app-operator

  ```
  - apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - clientregistrations
  verbs:
    - "*"
  ```

- app-editor

  ```
  - apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - clientregistrations
  verbs:
    - get
    - list
    - watch
  ```

- app-viewer

  ```
  - apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - clientregistrations
  verbs:
    - get
    - list
    - watch
  ```

- service-operator

  ```
  - apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - authserver
  verbs:
    - "*"
  ```

To manage the life cycle of AppSSO's APIs, the AppSSO controller's `ServiceAccount` has a `ClusterRole` with the following permissions:

```
- apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - authservers
  verbs:
    - get
```

```
    - list
    - watch
- apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - authservers/status
  verbs:
    - patch
    - update
- apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - authservers/finalizers
  verbs:
    - "*"
- apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - clientregistrations
  verbs:
    - get
    - list
    - watch
- apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - clientregistrations/status
  verbs:
    - patch
    - update
- apiGroups:
    - sso.apps.tanzu.vmware.com
  resources:
    - clientregistrations/finalizers
  verbs:
    - "*"
- apiGroups:
    - ""
  resources:
    - secrets
    - configmaps
    - services
    - serviceaccounts
  verbs:
    - "*"
- apiGroups:
    - apps
  resources:
    - deployments
  verbs:
    - "*"
- apiGroups:
    - rbac.authorization.k8s.io
  resources:
    - roles
    - rolebindings
  verbs:
    - "*"
- apiGroups:
    - cert-manager.io
  resources:
    - certificates
    - issuers
  verbs:
    - "*"
- apiGroups:
```

```
    - cert-manager.io
  resources:
    - clusterissuers
  verbs:
    - get
    - list
    - watch
    - apiGroups:
        - networking.k8s.io
      resources:
        - ingresses
      verbs:
        - "*"
- apiGroups:
    - ""
  resources:
    - events
  verbs:
    - create
    - update
    - patch
- apiGroups:
    - coordination.k8s.io
  resources:
    - leases
  verbs:
    - create
    - get
    - update
```

AppSSO installs *OpenShift*-specific RBAC and resources.

Application Single Sign-On for OpenShift clusters

On OpenShift clusters, AppSSO must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes Pod Security Standards. Tanzu Application Platform configures the following SCC for AppSSO controller and its `AuthServer` managed resources when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: appsso-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - KILL
  - MKNOD
  - SETUID
  - SETGID
```

```
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
seccompProfiles:
  - 'runtime/default'
```

AppSSO controller's `ServiceAccount` is given the following additional permissions, including a `use` permission for AppSSO SCC, so `AuthServer` can use the custom SCC:

```
- apiGroups:
    - security.openshift.io
  resources:
    - securitycontextconstraints
  verbs:
    - "get"
    - "list"
    - "watch"
```

```
- apiGroups:
    - security.openshift.io
  resourceNames:
    - appsso-scc
  resources:
    - securitycontextconstraints
  verbs:
    - "use"
```

Upgrade Application Single Sign-On

This topic tells you how to upgrade Application Single Sign-On (commonly called AppSSO) outside of a Tanzu Application Platform profile installation. If you installed Tanzu Application Platform through a profile, see Upgrade Tanzu Application Platform instead.

For help on migrating your resources in between versions, see the migration guides.

If you installed the `AppSSO` package on its own, and not as part of `TAP`, you can upgrade it individually by running:

```
tanzu package installed update PACKAGE-INSTALLATION-NAME -p sso.apps.tanzu.vmware.com
-v 3.1 --values-file PATH-TO-YOUR-VALUES-YAML -n YOUR-INSTALL-NAMESPACE
```

Note

You can also upgrade AppSSO as part of upgrading Tanzu Application Platform as a whole. See Upgrading Tanzu Application Platform for more information.

Migration guides

`v3.0.0` to `v3.1.0`

VMware recommends that you recreate your `AuthServers` after upgrading your AppSSO to `v3.1.0` with the following changes:

- Migrate field `.spec.identityProviders[*].openid.claimMappings["roles"]` to `.spec.identityProviders[*].openid.roles.fromUpstream.claim`.
- Migrate field `.spec.identityProviders[*].ldap.group.roleAttribute` to `.spec.identityProviders[*].ldap.roles.fromUpstream.attribute`.
- Migrate field `.spec.identityProviders[*].ldap.group.search` to `.spec.identityProviders[*].ldap.roles.fromUpstream.search`.
- Migrate field `.spec.identityProviders[*].saml.claimMappings["roles"]` to `.spec.identityProviders[*].saml.roles.fromUpstream.attribute`.

(Optional) If you plan to run Spring Boot 3 based `Workload`s, you must perform the following migration tasks in your existing `ClientRegistration` resources:

- Migrate `.spec.clientAuthenticationMethod` values.
- Migrate existing value `post` to `client_secret_post` or migrate existing value `basic` to `client_secret_basic`.

`v2.0.0` to `v3.0.0`

VMware recommends that you recreate your `AuthServers` after upgrading your AppSSO to `v3.0.0` with the following changes:

- Migrate field `.spec.tls.disabled` to `.spec.tls.deactivated`.

`v1.0.0` to `v2.0.0`

VMware recommends that you recreate your `AuthServers` after upgrading your AppSSO to `v2.0.0` with the following changes:

- Migrate from `.spec.issuerURI` to `.spec.tls`:

  Note

  AppSSO templates your issuer URI and enables TLS. When using the newer `.spec.tls`, a custom `Service` and an ingress resource are no longer required.

  It is not recommended to continue using `.spec.issuerURI` in AppSSO v2.0.0. To use `.spec.issuerURI` in AppSSO v2.0.0, you must provide a `Service` and an ingress resource as in AppSSO v1.0.0.

  1. Configure one of `.spec.tls.{issuerRef, certificateRef, secretRef}`. See Issuer URI & TLS for more information.
  2. (Optional) Disable TLS with `.spec.tls.disabled`.
  3. Remove `.spec.issuerURI`.
  4. Delete your `AuthServer`-specific `Service` and ingress resources.
  5. Apply your `AuthServer`. You can find its issuer URI in `.status.issuerURI`.
  6. Update the redirect URIs in your upstream identity providers.

- If you use the `internalUnsafe` identity provider to migrate existing users by replacing the bcrypt hash through the plain-text equivalent. You can still use existing bcrypt passwords by prefixing them with `{bcrypt}`:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  # ...
spec:
  identityProviders:
    - name: internal
      internalUnsafe:
        users:
          # v1.0
          - username: test-user-1
            password: $2a$10$201z9o/tHlocFsHFTo0plukh03ApBYe4dRiXcqeyRQH6CNNtS8
jWK # bcrypt-encoded "password"
            # ...
          # v2.0
          - username: "test-user-1"
            password: "{bcrypt}$2a$10$201z9o/tHlocFsHFTo0plukh03ApBYe4dRiXcqeyR
QH6CNNtS8jWK" # same bcrypt hash, with {bcrypt} prefix
          - username: "test-user-2"
            password: "password" # plain text
  # ...
```

Uninstall Application Single Sign-On

This topic tells you how to uninstall Application Single Sign-On (commonly called AppSSO).

Delete the AppSSO package by running:

```
tanzu package installed delete appsso --namespace tap-install
```

To permanently delete and exclude AppSSO package from your Tanzu Application Platform install, edit your Tanzu Application Platform values file by including the following configuration:

```
excluded_packages:
  - sso.apps.tanzu.vmware.com
```

For more information, navigate to Exclude packages from a Tanzu Application Platform profile.

Application Single Sign-On for Service Operators

The following topics tell you how to configure a fully operational authorization server for Application Single Sign-On (commonly called AppSSO):

- Annotations and labels
- Issuer URI and TLS
- TLS scenario guides
- CA certificates
- Configure Workloads to trust a custom CA
- Identity providers
- Configure authorization
- Public clients and CORS
- Token signatures
- Storage

- AuthServer readiness
- Scale AuthServer
- AuthServer audit logs

`AuthServer` represents the request for an OIDC authorization server. It results in the deployment of an authorization server backed by Redis over mutual TLS if no external storage is explicitly configured.

You can configure the labels with which clients can select an `AuthServer`, the namespaces it allows clients from, its issuer URI, its token signature keys, identity providers and further details for its deployment.

For the full available configuration, `spec` and `status` see the API reference.

Application Single Sign-On for Service Operators

The following topics tell you how to configure a fully operational authorization server for Application Single Sign-On (commonly called AppSSO):

- Annotations and labels
- Issuer URI and TLS
- TLS scenario guides
- CA certificates
- Configure Workloads to trust a custom CA
- Identity providers
- Configure authorization
- Public clients and CORS
- Token signatures
- Storage
- AuthServer readiness
- Scale AuthServer
- AuthServer audit logs

`AuthServer` represents the request for an OIDC authorization server. It results in the deployment of an authorization server backed by Redis over mutual TLS if no external storage is explicitly configured.

You can configure the labels with which clients can select an `AuthServer`, the namespaces it allows clients from, its issuer URI, its token signature keys, identity providers and further details for its deployment.

For the full available configuration, `spec` and `status` see the API reference.

Annotations and labels for AppSSO

This topic tells you how to configure annotations and labels for Application Single Sign-On (commonly called AppSSO).

An `AuthServer` is selectable by `ClientRegistration` through labels. The namespace an `AuthServer` allows `ClientRegistrations` from is controlled with an annotation.

Labels

`ClientRegistrations` select an `AuthServer` with `spec.authServerSelector`. Therefore, an `AuthServer` must have a set of labels that uniquely identifies it amongst all `AuthServer`. A `ClientRegistration` must match only one `AuthServer`. Registration fails if multiple or no `AuthServer` resources are matched.

For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  labels:
    env: dev
    ldap: True
    saml: True
# ...
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  labels:
    env: prod
    saml: True
# ...
```

Allowing client namespaces

`AuthServer` controls which namespace it allows `ClientRegistrations` with the annotation:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
```

To allow `ClientRegistrations` from all or a restricted set of namespaces this annotation must be set. Its value is a comma-separated list of allowed Namespaces, e.g. `"app-team-red,app-team-green"`, or `"*"` if it should allow clients from all namespaces.

Caution

If the annotation is missing, no clients are allowed.

Unsafe configuration

`AuthServer` is designed to enforce secure and production-ready configuration. However, sometimes it is necessary to opt-out of those constraints, e.g. when deploying `AuthServer` on an *iterate* cluster.

Caution

Allowing **unsafe** is not recommended for production.

Unsafe identity provider

The `InternalUnsafe` identity provider cannot be used unless explicitly allowed by including the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider` as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
spec:
  identityProviders:
    - name: static-users
      internalUnsafe:
      # ...
```

If the annotation is not present and an `InternalUnsafe` identity provider is configured the `AuthServer` will not apply.

Unsafe issuer URI

It's not possible to use a plain HTTP issuer URI, unless it's explicitly allowed by including the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri` as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""
spec:
  issuerURI: http://this.is.unsafe
```

If the annotation is not present and a plain HTTP issuer URI is configured, the `AuthServer` does not apply.

Issuer URI and TLS for AppSSO

This topic tells you how to configure the issuer URI and TLS for Application Single Sign-On (commonly called AppSSO).

Overview

An `AuthServer` entry point for its clients and their end-users is called *issuer URI*. AppSSO will template the issuer URI and create a TLS-enabled `Ingress` for it. For this purpose, your platform operator configures the domain name and template. Once you created and `AuthServer` you can find the actual URL in `.status.issuerURI`.

You can configure whether and how to obtain a TLS certificate for the issuer URI by using `.spec.tls`. Unless TLS is deactivated, HTTPS is enforced. For example, requests for `http://` are redirected to `https://`. You can observe the TLS configuration in `.status.tls`.

If AppSSO is installed with a default issuer, you can omit `AuthServer.spec.tls` and a TLS certificate is obtained automatically. This is the recommended approach for TLS.

For example:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
```

```
  namespace: services
  # ...
spec:
# ...
status:
  issuerURI: https://login.services.example.com
  tls:
    issuerRef:
      name: my-default-issuer
      kind: ClusterIssuer
      group: cert-manager.io
  # ...
```

This `AuthServer` is reachable at its templated issuer URI `https://login.services.example.com` and serves a TLS certificate obtained from `my-default-issuer`.

Learn how to configure TLS for your `AuthServer`:

- Configure TLS by using a (Cluster)Issuer
- Configure TLS by using a Certificate
- Configure TLS by using a Secret
- Deactivate TLS

There are many use-cases that pertain to TLS use. To find out which scenario applies to you and how to configure it, see TLS scenario guides.

If your `AuthServer` obtains a certificate from a custom CA, you can enable App Operators to trust it. See Allow Workloads to trust a custom CA AuthServer for more information.

Configure TLS by using a (Cluster)Issuer

You can obtain a TLS certificate for your `AuthServer` by referencing a `cert-manager.io/v1/Issuer` or `cert-manager.io/v1/ClusterIssuer`. This enables AppSSO to retrieve a `cert-manager.io/v1/Certificate` from the issuer and apply it to the `Ingress` configuration.

The composition of an `AuthServer` and a self-signed `Issuer` looks as follows:

```
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: my-selfsigned-issuer
  namespace: services
spec:
  selfSigned: { }

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    issuerRef:
      name: my-selfsigned-issuer
      # 'kind: Issuer' can be omitted. It is the default.
```

The composition of an `AuthServer` and a *self-signed* `ClusterIssuer` for looks as follows:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: my-selfsigned-cluster-issuer
spec:
  selfSigned: { }

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    issuerRef:
      name: my-selfsigned-cluster-issuer
      kind: ClusterIssuer
```

Confirm that your `AuthServer` serves a TLS certificate from the specified issuer by visiting its `{.status.issuerURI}`.

For more information about cert-manager and its APIs. see cert-manager documentation.

Configure TLS by using a Certificate

In order to configure TLS for your `AuthServer` using a `cert-manager.io/v1/Certificate` you must know what its templated issuer URI will be. You can infer it from the AppSSO package's domain template.

The composition of an `AuthServer` and a `Certificate` looks as follows:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: login
  namespace: services
spec:
  dnsNames:
    - login.services.example.com
  issuerRef:
    name: my-cluster-issuer
    kind: ClusterIssuer
  secretName: login-cert

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    certificateRef:
      name: login
```

Confirm that your `AuthServer` serves the specified Certificate by visiting its `{.status.issuerURI}`.

For more information about cert-manager and its APIs. see cert-manager documentation.

Configure TLS by using a Secret

If you don't want to use cert-manager.io's APIs or you have a raw TLS certificate in a TLS `Secret`, you can compose it with your `AuthServer` by referencing it. The certificate must be for the issuer URI that will be templated for the `AuthServer`. You can infer it from the AppSSO package's domain template.

The composition of an `AuthServer` and TLS `Secret` looks as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-tls-cert
  namespace: services
type: kubernetes.io/tls
data:
  tls.key: # ...
  tls.crt: # ...

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  tls:
    secretRef:
      name: my-tls-cert
```

Deactivate TLS (unsafe)

If you deactivate TLS autoconfiguration, `AuthServer` only works over plain HTTP. You must deactivate TLS with the `sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""` annotation.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""
  # ...
spec:
  tls:
    deactivated: true
```


Caution

Deactivating TLS is unsafe and not recommended for production.

Allow `Workloads` to trust a custom CA `AuthServer`

If your `AuthServer` obtains a certificate from a custom CA, its consumers do not trust it by default. You can enable App Operators' `Workloads` to trust your `AuthServer` by exporting a `ca-certificates` service binding `Secret` to their `Namespace`.

A composition of `SecretTemplate` and `SecretExport` are a way to achieve this. If your custom CA's TLS `Secret` is present in the namespace `my-certs`, then you can provide a `ca-certificates` service binding `Secret` like so:

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretTemplate
metadata:
  name: ca-cert
  namespace: my-certs
spec:
  inputResources:
    - name: my-custom-ca
      ref:
        apiVersion: v1
        kind: Secret
        name: my-custom-ca
  template:
    data:
      ca.crt: $(.my-custom-ca.data.tls\.crt)
    stringData:
      type: ca-certificates

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: ca-cert
  namespace: my-certs
spec:
  toNamespace: "*"
```

This templates a `ca-certificates` service binding `Secret` which `Workload` can claim to trust the custom CA. It does not contain the CA's private key and is generally safe to share.

However, be careful, this example exports to all namespace on the cluster. If this does not comply with your policies, then adjust the target namespaces if required.

For more information about secretgen-controller and its APIs, see secretgen-controller documentation in GitHub.

TLS scenario guides for AppSSO

This topic tells you how to obtain a TLS certificate in different scenarios for Application Single Sign-On (commonly called AppSSO).

Overview

`AuthServer` is a piece of security infrastructure. It is imperative to configure TLS for it, so that its issuer URI's scheme is `https://`.

`AuthServer.spec.tls` accommodates different scenarios for obtaining a TLS certificate. Select the scenario that matches your case.

The recommended path is to install AppSSO with a default issuer. In that case, you can omit `AuthServer.spec.tls` and a TLS certificate is obtained automatically.

Prerequisites

Each of the scenarios requires that the AppSSO package is installed and configured. In particular, its `domain_name` must match the ingress domain of your cluster. The presented YAML resources

assume `my-tap.example.com` as the ingress domain. Therefore, the AppSSO configuration values look as follows:

```
#! AppSSO values
domain_name: "my-tap.example.com"
```

The default `domain_template: "{{.Name}}.{{.Namespace}}.{{.Domain}}"` works for most scenarios. If a scenario requires a bespoke `domain_template`, it contains the relevant instructions.

After applying each scenario, wait for your `AuthServer` to become ready and then test it by running:

```
kubectl wait --namespace login authserver/sso --for condition=Ready=True --timeout 500
s
curl --location "$(kubectl get --namespace login authserver sso --output=jsonpath='{.s
tatus.issuerURI}')/.well-known/openid-configuration"
```

Alternatively, visit the `AuthServer` with your browser. You can obtain its issuer URI by running:

```
kubectl get --namespace login authserver sso --output=jsonpath='{.status.issuerURI}'
```


Caution

Before applying each scenario, you must configure your AppSSO correctly, and make sure that all certificates and DNS names comply with your setup.

Using a default issuer

VMware recommend using a default issuer,
because this approach separates the responsibilities of platform operators and service operators. In this case, the `Authserver.spec.tls` field is not required.

To verify whether `AppSSO` was installed with a default issuer, run:

```
kctrl package installed get --namespace tap-install --package-install tap --values-fil
e-output tap-values.yaml
```

If a `shared.ingress_issuer` appears in your `tap-values.yaml` file, you have a default issuer.

Important

Ensure kctrl is installed.

```
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
    sso.apps.tanzu.vmware.com/documentation: Uses the default issuer for TLS
```

```
spec:
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Using a `ClusterIssuer`

A `ClusterIssuer` is a cluster-scoped API provided by cert-manager from which certificates can be obtained programmatically.

This scenario puts all resources into a single YAML file and uses Let's Encrypt's production API. You might get the `ClusterIssuer` from your platform operators.

For more information, see cert-manager documentation.

Caution

Let's Encrypt's production API rate limits apply.

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-production
spec:
  acme:
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
```

```
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    issuerRef:
      name: letsencrypt-production
      kind: ClusterIssuer
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Using an `Issuer`

This scenario is identical to Using a ClusterIssuer, except that the Issuer is scoped to a namespace and must be located in the same namespace as the `AuthServer`.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: letsencrypt-production
  namespace: login
spec:
  acme:
    privateKeySecretRef:
      name: letsencrypt-production
    server: https://acme-v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: contour

---
```

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    issuerRef:
      name: letsencrypt-production
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Using an existing `Certificate`

A `Certificate` is an API provided by cert-manager that is scoped to a namespace and represents a TLS certificate obtained from a `(Cluster)Issuer`. To create a `Certificate`, you must know the name and kind of your issuer.

These scenarios use Let's Encrypt's production API and require that a `ClusterIssuer` by the name `letsencrypt-production` exists. See Using a `ClusterIssuer` for how to set up the issuer.

When using `Certificate`, its `.spec.dnsNames` must contain the FQDN of the templated issuer URI. The `domain_name` and `domain_template` of your AppSSO package installation must comply with your DNS name.

If you have an existing `Certificate` in the same `Namespace` where the `AuthServer` is installed, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: sso
```

```
  namespace: login
spec:
  dnsNames:
    - "sso.login.my-tap.example.com"
  issuerRef:
    name: letsencrypt-production
    kind: ClusterIssuer
  secretName: sso-cert

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    certificateRef:
      name: sso
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

secretgen-controller allows you to export and import `Secrets` across namespaces. When your `Certificate` is located in another namespace, for example, it's controlled by another team, you can import its `Secret` to other namespaces. If you have an existing `Certificate` in another `Namespace`, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: tls

---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: sso
  namespace: tls
spec:
```

```
  dnsNames:
    - "sso.login.my-tap.example.com"
  issuerRef:
    name: letsencrypt-production
    kind: ClusterIssuer
  secretName: sso-cert

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: sso-cert
  namespace: tls
spec:
  toNamespace: login

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: sso-cert
  namespace: login
spec:
  fromNamespace: tls

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: sso-cert
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
```

```
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```


Caution

Be cautious when using `SecretExport` and `SecretImport` to facilitate the transfer across namespaces.

Using an existing TLS certificate

If you have an existing TLS certificate and private key, for example, if your TLS certificate was created outside the cluster, you can apply it directly.

If you don't have a TLS certificate, there are numerous ways to obtain TLS certificates. One of the simplest methods is to use a tool such as mkcert, step or openssl in GitHub.

If you have an existing TLS certificate in the same `Namespace` where the AuthServer is installed, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-cert
  namespace: login
stringData:
  #! --- ReplaceMe - certificate and private key for "sso.login.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! ---------------------------------------------------------------------------

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-cert
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
```

```
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Important

The TLS certificate `tls.crt` and its corresponding private key `tls.key` must be stored in a secret with these keys.

If you have an existing TLS certificate in another `Namespace`, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: tls

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-cert
  namespace: tls
stringData:
  #! --- ReplaceMe - certificate and private key for "sso.login.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! ---------------------------------------------------------------------------

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: my-cert
  namespace: tls
spec:
  toNamespace: login
```

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: my-cert
  namespace: login
spec:
  fromNamespace: tls

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-cert
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Important

The TLS certificate `tls.crt` and its corresponding private key `tls.key` must be stored in a secret with these keys.

Be cautious when using `SecretExport` and `SecretImport` to facilitate the transfer across namespaces.

Using an existing wildcard TLS certificate

To use wildcard certificates for DNS names such as `*.my-tap.example.com`, you must edit the AppSSO's `domain_template` so that the templated issuer URIs for `AuthServer` match the wildcard. For example:

- `sso.login.my-tap.example.com` does not match the wildcard.
- `sso-login.my-tap.example.com` matches the wildcard.

The following AppSSO configuration values accommodates a wildcard certificate for `*.my-tap.example.com`:

```
#! AppSSO values
domain_name: "my-tap.example.com"
domain_template: "{{.Name}}-{{.Namespace}}.{{.Domain}}"
#!                          ^ note the dash
```

The following scenarios require TLS Secrets, but the same concept applies to `Certificate`.

Important

When using a `(Cluster)Issuer` for Let's Encrypt, you cannot request wildcard certificates when it uses the http01 challenge solver.

If you have an existing wildcard TLS certificate in the same `Namespace` where the AuthServer is installed, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-wildcard-cert
  namespace: login
stringData:
  #! --- ReplaceMe - certificate and private key for "*.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! ---------------------------------------------------------------------------

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
```

```
  tls:
    secretRef:
      name: my-wildcard-cert
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Note

This scenario is similar to using an existing TLS certificate in the same namespace, except that the certificate is a wildcard.

If you have an existing wildcard TLS certificate in another `Namespace`, use the following AppSSO configuration values:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: tls

---
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: my-wildcard-cert
  namespace: login
stringData:
  #! --- Certificate and private key for "*.my-tap.example.com ---
  tls.crt: |
    -----BEGIN CERTIFICATE-----
    # redacted
    -----END CERTIFICATE-----
  tls.key: |
    -----BEGIN PRIVATE KEY-----
    # redacted
    -----END PRIVATE KEY-----
  #! ---------------------------------------------------------------

---
apiVersion: secretgen.carvel.dev/v1alpha1
```

```
kind: SecretExport
metadata:
  name: my-cert
  namespace: tls
spec:
  toNamespace: login

---
apiVersion: v1
kind: Namespace
metadata:
  name: login

---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: my-cert
  namespace: login
spec:
  fromNamespace: tls

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: sso
  namespace: login
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
spec:
  #! --- TLS ---
  tls:
    secretRef:
      name: my-wildcard-cert
  #! -----------
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: user
            password: password
  tokenSignature:
    signAndVerifyKeyRef:
      name: signing-key

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: signing-key
  namespace: login
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)
```

Note

This scenario is similar to using an existing TLS certificate in another namespace, except that the certificate is a wildcard.

CA certificates for AppSSO

This topic tells you how to configure CA certificates for `AuthServer` in Application Single Sign-On (commonly called AppSSO).

An `AuthServer` can trust custom CAs. You can establish either for all `AuthServer`s or for a single `AuthServer`. This is useful when either your identity provider or storage serves certificates from a custom CA.

In most cases, CA certificates are PEM-encoded and located in a `Secret` referred from `.spec.caCerts[].secretRef.name`. The controller considers all `Secret` entries matching `*.(crt|ca-bundle)`. That means you can include multiple CA certificates in a single `Secret` or spread them across multiple `Secret`s.

After being created, an `AuthServer` reports the trusted, total custom CA certificates in its `.status.caCerts` together with the location where it sources them from. This includes the CA certificates that are trusted by all `AuthServer`s.

For example:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: my-ca
  namespace: services
stringData:
  my.ca-bundle: |
    This is My Company's custom CA. It's common name is "My CA".
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: login
  namespace: services
  # ...
spec:
  caCerts:
    - secretRef:
        name: my-ca
  # ...
status:
  caCerts:
    - cert:
        subject: CN=My CA,O=My Company,C=Happyland
      source:
        secretEntry: service/my-ca[my.ca-bundle]
    - cert:
        subject: CN=My other CA,O=My Company,C=Happyland
      source:
        secretEntry: appsso/appsso-controller[controller.yaml]
  # ...
```

CA certificates configured for all `AuthServer`s by using the package installation's `ca_cert_data` are sourced from `secretEntry: appsso/appsso-controller[controller.yaml]`. This denotes the

AppSSO controller's configuration `Secret`.

Configure workloads to trust a custom CA

This topic tells you how to configure workloads to trust a custom Certificate Authority (commonly called CA) for Application Single Sign-On (commonly called AppSSO).

Overview

If your `ClientRegistration` selects an `AuthServer` that serves a certificate from a custom CA, your `Workload` does not trust it by default. This is because the certificate is not issued by a trusted certificate authority from the `Workload`'s perspective.

To establish trust between a `Workload` and an `AuthServer`:

Step	Task	Link
1.	Service Operator exports the custom CA certificate `Secret` resource from the namespace in which it is issued.	Exporting custom CA certificate Secret
2.	Service Operator imports the custom CA certificate `Secret` to the namespace in which the `Workload` is created.	Importing custom CA certificate Secret
3.	Append the deployed `Workload` as a service resource claim, denoting the custom CA certificate `Secret` in the workload namespace.	Appending custom CA certificate Secret reference to Workload

Important

These steps are mandatory if Tanzu Application Platform is installed with the default self-signed `ClusterIssuer` resource, in which the CA is custom.

Exporting custom CA certificate Secret

A `ca-certificates` service binding `Secret` allows to configure trust for custom CAs.

For more information about exporting CA certificate Secrets, see Allow Workloads to trust a custom CA AuthServer.

Example: Create a `ca-certificates`-type ServiceBinding Secret from template and offer Tanzu Application Platform's default self-signed CA certificate Secret to workloads namespace.

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretTemplate
metadata:
  name: tap-ca-cert
  namespace: cert-manager                    # The namespace in which your custom CA
Secret resides.
spec:
  inputResources:
    - name: tap-ingress-selfsigned-root-ca
      ref:
        apiVersion: v1                       # The custom CA certificate Secret.
        kind: Secret                         # ^^
        name: tap-ingress-selfsigned-root-ca  # ^^
  template:
    data:
      ca.crt: $(.tap-ingress-selfsigned-root-ca.data.tls\.crt)
    stringData:
      type: ca-certificates
```

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: tap-ca-cert           # The name of the SecretTemplate that created the "ca-ce
rtificates" Secret.
  namespace: cert-manager     # The namespace in which Tanzu Application Platform's se
lf-signed ClusterIssuer stores its CA cert Secret.
spec:
  toNamespace: my-apps        # The namespace in which Workloads are deployed.
```

Importing custom CA certificate Secret

After the custom CA certificate Secret is exported from its original namespace, you can import it into the workloads' namespace.

Example: Accept Tanzu Application Platform's default self-signed CA certificate Secret offer.

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretImport
metadata:
  name: tap-ca-cert
  namespace: my-apps            # The namespace in which Workloads are deployed.
spec:
  fromNamespace: cert-manager   # The namespace in which your custom CA certificate Se
cret resides.
```

Appending custom CA certificate Secret reference to Workload

With custom CA certificate available in the workloads' namespace, you can append it to the `Workload` as a service resource claim:

Example: Appending custom CA certificate Secret as a resource claim.

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
# ...
spec:
  serviceClaims:
    - name: ca-cert
      ref:
        apiVersion: v1    # The custom CA Secret template that is imported into the wo
rkloads' namespace.
        kind: Secret      # ^^
        name: tap-ca-cert # ^^
    # ...
```

Alternatively, you can provide the workload with a `--service-ref` parameter for the same effect:

```
--service-ref "ca-cert=v1:Secret:tap-ca-cert"
```

For more information about secretgen-controller and its APIs, see secretgen-controller documentation in GitHub.

Identity providers for AppSSO

This topic tells you how to configure Application Single Sign-On (commonly called AppSSO) to use external identity providers (commonly called IdPs).

Users can log in by using external identity providers. OpenID Connect and LDAP providers are supported. SAML providers have limited experimental support. An `AuthServer` does not manage users internally. Developers can get started quickly without needing to connect to an IdP by using static hard-coded users, which is for development purposes only.

Identity providers are configured under `spec.identityProviders`, learn more from the API reference.

> **Caution**
>
> Changes to `spec.identityProviders` do not take effect immediately because the operator will roll out a new deployment of the authorization server.

End-users will be able to log in with these providers when they go to `{spec.issuerURI}` in their browser.

Learn how to configure identity providers for an `AuthServer`:

- OpenID
- LDAP
- SAML (experimental)
- Internal, static user
- Roles claim filtering
- Roles claim mapping and filtering explained
- Configure authorization
- Restrictions

OpenID Connect providers

To set up an OpenID Connect provider, provide the following information for your `AuthServer`:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: my-oidc-provider
      openID:
        issuerURI: https://openid.example.com
        clientID: my-client-abcdef
        clientSecretRef:
          name: my-openid-client-secret
        scopes:
          - "openid"
          - "other-scope"
        authorizationUri: https://example.com/oauth2/authorize
        tokenUri: https://example.com/oauth2/token
        jwksUri: https://example.com/oauth2/jwks
        roles:
          fromUpstream:
            claim: "my-oidc-provider-groups"
  # ...
---
```

```
apiVersion: v1
kind: Secret
metadata:
  name: my-openid-client-secret
  # ...
stringData:
  clientSecret: very-secr3t
```

Where:

- `openID` is the issuer identifier. You can define as many OpenID providers as you like. If the provider supports OpenID Connect Discovery, the value of `openID` is used to auto-configure the provider by using information from `https://openid.example.com/.well-known/openid-configuration`.
- The value of `issuerURI` must not contain `.well-known/openid-configuration` and must match the value of the `issuer` field. See OpenID Connect documentation at `https://openid.example.com/.well-known/openid-configuration` for more information.

  Note

  You can retrieve the values of `issuerURI` and `clientID` when registering a client with the provider, which in most cases, is by using a web UI.

- `scopes` is used in the authorization request. Its value must contain `"openid"`. Other common `OpenID` values include `"profile"` and `"email"`. You can also run `curl -s "https://openid.example.com/.well-known/openid-configuration" | jq -r ".issuer"` to retrieve the correct `issuerURI` value.
- The value of `clientSecretRef` must be a `Secret` with the entry `clientSecret`.
- `authorizationUri` (optional) is the URI for performing an authorization request and obtaining an `authorization_code`.
- `tokenUri` (optional) is the URI for performing a token request and obtaining a token.
- `jwksUri` (optional) is the JSON Web Key Set (JWKS) endpoint for obtaining the JSON Web Keys to verify token signatures.
- `roles.fromUpstream.claim` (optional) selects which claim in the `id_token` contains the `roles` of the user. `roles` is a non-standard OpenID Connect claim. When `ClientRegistrations` has a `roles` scope, it is used to populate the `roles` claim in the `id_token` issued by the `AuthServer`. For more information, see OpenID external groups mapping.
  - `my-oidc-provider-groups` claim from the ID token issued by `my-oidc-provider` is mapped into the `roles` claim in id tokens issued by AppSSO.

Verify the configuration by visiting the `AuthServer`'s issuer URI in your browser and select `my-oidc-provider`.

OpenID external groups mapping

Service operators may map the identity provider's "groups" (or equivalent) claim to the `roles` claim within an `AuthServer`'s identity token.

Note

Read more about roles claim mapping and filtering here

App Operators may configure their ClientRegistration to have the `roles` claim included in the `id_token`.

Configure `AuthServer` with OpenID Connect groups mapping:

```
spec:
  identityProviders:
    - name: "openid-idp"
      openid:
        scopes:
          - upstream-identity-providers-groups-claim # Optional based on the identity
provider.
        roles:
          fromUpstream:
            claim: "upstream-identity-providers-groups-claim"
```


Caution

Some OpenID providers, such as Okta OpenID, might require requesting the roles or groups scope from the identity provider, as a result, you must include it in the `.openid.scopes` list.

For every ClientRegistration that has the `roles` scope listed, the identity token will be populated with the `roles` claim:

```
kind: ClientRegistration
metadata:
  name: my-client-registration
spec:
  scopes:
    - name: openid
    - name: roles
  # ...
```

When groups are mapped (as described above), all the groups provided by the identity provider are retrieved, and the relevant groups that the logged-in user is part of are appended to the `roles` claim of an `id_token`. To filter the available roles within an `id_token`, see Roles claim filtering section.

Note for registering a client with the identity provider

The `AuthServer` will set up redirect URIs based on the provider name in the configuration. For example, for a provider with `name: my-provider`, the redirect URI will be `{spec.issuerURI}/login/oauth2/code/my-provider`. The externally accessible user URI for the `AuthServer`, including scheme and port is `spec.issuerURI`. If the `AuthServer` is accessible on `https://appsso.company.example.com:1234/`, the redirect URI registered with the identity provider should be `https://appsso.company.example.com:1234/login/oauth2/code/my-provider`.

Supported token signing algorithms

AppSSO only supports the `RS256` algorithm for token signature. For more information, see OpenID Connect documentation.

You can find out the signing algorithms your OpenID provider supports by referring to the `id_token_signing_alg_values_supported` response parameter in the OpenID Connect documentation at `.well-known/openid-configuration`.

For example, you can run:

```
curl -s "ISSUER-URI/.well-known/openid-configuration" | jq ".id_token_signing_alg_valu
es_supported"
```

LDAP

Important

You can not configure more than one `ldap` identity provider.

For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        url: "ldaps://example.com:636"
        bind:
          dn: uid=binduser,ou=Users,dc=example,dc=com
          passwordRef:
            name: ldap-password
        user:
          searchBase: ou=Users,dc=example,dc=com
          searchFilter: cn={0}
        roles:
          fromUpstream:
            attribute: cn
            search:
              filter: member={0}
              base: ou=Users,dc=example,dc=com
              searchSubTree: true
              depth: 10
          filterBy:
            - exactMatch: "users"
            - regex: "^admin"
        group: # DEPRECATED, use 'roles' instead
          search:
            filter: member={0}
            base: ou=Users,dc=example,dc=com
            searchSubTree: true
            depth: 10
          roleAttribute: cn
  # ...
---
apiVersion: v1
kind: Secret
metadata:
  name: ldap-password
  namespace: default
stringData:
  password: confidential-password-value
```

Where:

- `url` is the URL of the LDAP server. It must be `ldaps` and must contain a port.
- `bind.dn` is the DN to perform the bind.

- `bind.passwordRef` must be a secret with the entry `password`. That entry is the password to perform the bind.
- `user.searchBase` is the branch of tree where the users are located at. Search is performed in nested entries.
- `user.seachFilter` is the filter for LDAP search. It must contain the string `{0}`, which is replaced by the `dn` of the user when performing a search. For example, when logging in with the username `marie`, the filter for LDAP search is `cn=marie`.
- `roles` (optional) defaults to unset. It configures how LDAP groups are mapped to user roles in the `id_token` claims. If not set, the user has no roles.
    - `fromUpstream.attribute` selects which attribute of the group entry are mapped to a user role. If an attribute has multiple values, the first value is selected.
    - `fromUpstream.search` (optional) toggles “OpenLDAP”-style group search, optionally uses recursive search to find groups for a given user.
    - `filterBy` (optional) - applied roles claim filters. See Roles claim filtering section for more details.
- `group` (DEPRECATED, , use `role` instead, optional) defaults to unset. It configures how LDAP groups are mapped to user roles in the `id_token` claims. If not set, the user has no roles.
    - `group.roleAttribute` selects which attribute of the group entry are mapped to a user role. If an attribute has multiple values, the first value is selected.
    - `group.search` (optional) toggles “Active Directory” search and uses recursive search to find groups for a given user.

Verify the configuration by visiting the `AuthServer`’s issuer URI in your browser and log in with the username and password from LDAP.

LDAP external groups mapping

Service operators may map the identity provider’s “groups” (or equivalent) attribute to the `roles` claim within an `AuthServer`’s identity token.

Note

Read more about roles claim mapping and filtering here

Configure `AuthServer` with LDAP groups attribute mapping:

```
spec:
  identityProviders:
    - name: "ldap-idp"
      ldap:
        roles:
          fromUpstream:
            attribute: "upstream-identity-providers-groups-attribute" # e.g. "cn" or
"dn"
```

For every ClientRegistration that has the `roles` scope listed, the identity token will be populated with the `roles` claim:

```
kind: ClientRegistration
metadata:
  name: my-client-registration
spec:
```

```
  scopes:
    - name: openid
    - name: roles
  # ...
```

When groups are mapped (as described above), all the groups provided by the identity provider are retrieved, and the relevant groups that the logged-in user is part of are appended to the `roles` claim of an `id_token`. To filter the available roles within an `id_token`, see Roles claim filtering section.

ActiveDirectory group search

In ActiveDirectory groups, user entries have a multi-value `memberOf` attribute, which contains the DNs pointing to the groups of a particular user. To enable this search mode, make sure `roles.fromUpstream.attribute` is set and `roles.fromUpstream.search` is not set.

For example:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        user:
          searchBase: OU=Cloud,DC=ad,DC=example,DC=com
          searchFilter: cn={0}
        roles:
          fromUpstream:
            attribute: sAMAccountName
```

The LDIF definition is as follows:

```
dn: CN=appsso-user,OU=Cloud,DC=ad,DC=example,DC=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: appsso user
sn: user
givenName: appsso
distinguishedName: CN=appsso user,OU=Cloud,DC=ad,DC=example,DC=com
displayName: appsso user
memberOf: CN=ssogroup,OU=Cloud,DC=ad,DC=example,DC=com
memberOf: CN=developers,OU=Cloud,DC=ad,DC=example,DC=com
sAMAccountName: appssouser
userPrincipalName: appssouser@ad.example.com
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=ad,DC=example,DC=com
# ...

# Groups
dn: CN=ssogroup,OU=Cloud,DC=ad,DC=example,DC=com
objectClass: top
objectClass: group
cn: ssogroup
member: CN=appsso-user,OU=Cloud,DC=ad,DC=example,DC=com
sAMAccountName: SSO Group
# ...

dn: CN=developers,OU=Cloud,DC=ad,DC=example,DC=com
objectClass: top
```

```
objectClass: group
cn: developers
sAMAccountName: Developers
# ...
```

The user `appsso-user` has two values for `memberOf`, pointing to two groups. Given the configuration earlier, `sAMAccountName` is used for the role, so the user has `SSO Group` and `Developers` as roles. The group is not required to have `member` attribute point to the user for the role to be mapped.

“Classic” group search

In non-ActiveDirectory LDAP, users generally do not have a `memberOf` attribute. Group search is performed by looking up groups in a base branch and filtering based on the groups `member` attribute.

An AuthServer can optionally perform:

- group search in sub-branches.
- nested group search, that is, find a hierarchy of groups, in which a group is a member of another group.

The complete configuration is as follows:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        roles:
          fromUpstream:
            attribute: description
            search:
              base: ou=Users,dc=example,dc=com
              filter: member={0}
              depth: 10
              searchSubTree: true
  # ...
```

Where:

- `search.base` is the base for running an LDAP search for groups.
- `search.filter` is the filter for running an LDAP search for groups. It must contain the string `{0}`, which is replaced by the `dn` of the user when performing group search. For example, `member=cn=marie,ou=Users,dc=example,dc=org`.
- `search.depth` (optional) is the depth at which to perform nested group search. It defaults to unset if left empty.
- `search.searchSubTree` (optional) controls whether to look for groups in sub-trees of the `search.base`. It defaults to unset if left empty.

Direct group search only

Given the following minimal configuration:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
```

```
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        user:
          searchBase: ou=Users,dc=example,dc=com
          searchFilter: uid={0}
        roles:
          fromUpstream:
            attribute: description
            search:
              base: ou=Users,dc=example,dc=com
              filter: member={0}
  # ...
```

The LDIF definition is as follows:

```
######################
## Users
######################
## User Marie Curie
## Marie Salomea Skłodowska Curie ; https://en.wikipedia.org/wiki/Marie_Curie
dn: cn=marie,ou=Users,dc=example,dc=org
cn: Marie
sn: Skłodowska Curie
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: marie

######################
## Groups
######################
dn: cn=nobels,ou=Users,dc=example,dc=org
objectClass: groupOfNames
description: Nobel Prizes
member: cn=marie,ou=Users,dc=example,dc=org
```

User `marie` has roles `Nobel Prizes`.

Groups in sub-trees

AppSSO can perform group search in sub-trees of the base for group search. This is enabled when `roles.fromUpstream.search.searchSubTree` is explicitly set to `true`. For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        roles:
          fromUpstream:
            attribute: description
            search:
              base: ou=Users,dc=example,dc=com
```

```
              filter: member={0}
              searchSubTree: true
  # ...
```

The LDIF definition is as follows:

```
######################
## Users
######################
## User Corazon
## Maria Corazon Sumulong Cojuangco Aquino; https://en.wikipedia.org/wiki/Corazon_Aqui
no
dn: cn=corazon,ou=Users,dc=example,dc=com
cn: Maria Corazon
sn: Sumulong Cojuangco Aquino
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: corazon


######################
## Groups
######################
dn: cn=presidents,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Presidents
member: cn=corazon,ou=Users,dc=example,dc=com

dn: cn=chief-commanders,ou=LegionHonor,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Chief Commanders
member: cn=corazon,ou=Users,dc=example,dc=com
```

User `corazon` has roles `Presidents` and `Chief Commanders`, which are retrieved from `ou=LegionHonor,ou=Users,dc=example,dc=com`, a subtree of the search base.

Nested group search

AppSSO can perform nested group search by going up a chain where a user is a member of a group, which is itself a member of a group, and so on. This is enabled by setting `roles.fromUpstream.search.depth` to greater than `1`. `roles.fromUpstream.search.depth` controls the number of “levels” that AppSSO fetches to get the groups of a user.

For example:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  identityProviders:
    - name: ldap
      ldap:
        # ...
        user:
          searchBase: ou=Users,dc=example,dc=com
          searchFilter: uid={0}
        roles:
          fromUpstream:
            attribute: description
            search:
              base: ou=Users,dc=example,dc=com
```

```
              filter: member={0}
              depth: 2
  # ...
```

The LDIF definition is as follows:

```
######################
## Users
######################
## User Corazon
## Maria Corazon Sumulong Cojuangco Aquino; https://en.wikipedia.org/wiki/Corazon_Aqui
no
dn: cn=corazon,ou=Users,dc=example,dc=com
cn: Maria Corazon
sn: Sumulong Cojuangco Aquino
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: corazon


######################
## Groups
######################
# Citizen > Politicians > Presidents > corazon (depth = 3)
dn: cn=citizens,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Citizens
member: cn=politicians,ou=Users,dc=example,dc=com

# Politicians > Presidents > corazon (depth = 2)
dn: cn=politicians,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Politicians
member: cn=presidents,ou=Users,dc=example,dc=com

# Presidents > corazon (depth = 1, direct group)
dn: cn=presidents,ou=Users,dc=example,dc=com
objectClass: groupOfNames
description: Presidents
member: cn=corazon,ou=Users,dc=example,dc=com
```

User `corazon` has roles `Presidents` and `Politicians`. However, the search stops at depth 2, so they do not have the role `Citizens`, which requires a depth greater or equal to 3.

SAML (experimental)

Caution

Support for SAML providers is experimental.

For SAML providers only autoconfiguration through `metadataURI` is supported.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
# ...
spec:
  - name: my-saml-provider
    saml:
      metadataURI: https://saml.example.com/sso/saml/metadata # required
```

```
    claimMappings: # optional
      # Map SAML attributes into claims in id_tokens issued by AppSSO. The key
      # on the left represents the claim, the value on the right the attribute.
      givenName: FirstName
      familyName: LastName
      emailAddress: email
```

SAML external groups mapping

Service operators may map the identity provider's "groups" (or equivalent) attribute to the `roles` claim within an `AuthServer`'s identity token.

Note

Read more about roles claim mapping and filtering here

Configure `AuthServer` with SAML role attribute:

```
spec:
  identityProviders:
    - name: "saml-idp"
      saml:
        roles:
          fromUpstream:
            attribute: "saml-group-attribute"
```

For every ClientRegistration that has the `roles` scope listed, the identity token will be populated with the `roles` claim:

```
kind: ClientRegistration
metadata:
  name: my-client-registration
spec:
  scopes:
    - name: openid
    - name: roles
  # ...
```

When groups are mapped (as described above), all the groups provided by the identity provider are retrieved, and the relevant groups that the logged-in user is part of are appended to the `roles` claim of an `id_token`. To filter the available roles within an `id_token`, see Roles claim filtering section.

Note for registering a client with the identity provider

The `AuthServer` will set up SSO and metadata URLs based on the provider name in the configuration. For example, for a SAML provider with `name: my-provider`, the SSO URL will be `{spec.issuerURI}/login/saml2/sso/my-provider`. The metadata URL will be `{spec.issuerURI}/saml2/service-provider-metadata/my-provider`. `spec.issuerURI` is the externally accessible issuer URI for an `AuthServer`, including scheme and port. If the `AuthServer` is accessible on `https://appsso.company.example.com:1234/`, the SSO URL registered with the identity provider should be `https://appsso.company.example.com:1234/login/saml2/sso/my-provider`.

Internal users

Caution

`InternalUnsafe` is unsafe and not recommended for production.

During development, static users can be useful for testing purposes. You can not configure more than one `internalUnsafe` identity provider.

For example:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
  # ...
spec:
  identityProviders:
    - name: test-users
      internalUnsafe:
        users:
          - username: ernie
            password: "password" # plain text
            roles:
              - "silly"
          - username: bert
            password: "{bcrypt}$2a$10$201z9o/tHlocFsHFTo0plukh03ApBYe4dRiXcqeyRQH6CNNt
S8jWK" # bcrypt-hashed "password"
            roles:
              - "grumpy"
  # ...
```

`InternalUnsafe` needs to be explicitly allowed by setting the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""`.

The passwords can be plain text or bcrypt-hashed. When bcrypt-hashing passwords they have to be prefixed with `{bcrypt}` . Learn how to bcrypt-hash string below.

Verify the configuration by visiting the `AuthServer`'s issuer URI in your browser and logging in as `ernie/password` or `bert/password`.

Generating a bcrypt hash from a plain-text password

There are multiple options for generating bcrypt hashes:

1. Use an online bcrypt generator
2. On Unix platforms, use `htpasswd`. Note, you may need to install it, for example on Ubuntu by running `apt install apache2-utils`

   ```
   htpasswd -bnBC 12 "" your-password-here | tr -d ':\n'
   ```

Roles claim filtering

Note

This section is applicable to OpenID, LDAP, and SAML (experimental) identity provider configurations.

Once an external groups mapping has been applied (as described per identity provider above), AppSSO is able to retrieve all the groups from an identity provider. An identity provider may have hundreds of groups, and any particular user may be part of a large subset of those groups. When a

user logs in, those groups will be appended to their `id_token`'s `roles` claim. This section describes how to filter the `roles` claim.

To filter groups, for a supported identity provider, apply:

```
spec:
  identityProviders:
    - name: my-provider
      <idp>:
        roles:
          filterBy:
            - exactMatch: ""
            - regex: ""
```

where `<idp>` may be `openid`, `ldap`, or `saml`.

Filters are disjunctive ("OR" operator), so each filter is applied to the entire set of groups, and merged into a set of unique filtered groups values. See filter examples for more information.

Roles claim filters

Available filters are:

- `exactMatch` - match the groups exactly. This filter is **case-sensitive**. e.g. `exactMatch: "developer"` will match only the group named "developer" and no other.
- `regex` - match groups according to the defined regular expression pattern. , and This filter is **case-insensitive**. e.g. `regex: ^admin` will match groups starting with the word "admin".
    - The regular expression pattern syntax used is RE2
    - Expressions should not be surrounded by forward slashes (`/`) and should only contain the pattern (e.g. `.*`, `^dev`, `\w+`).

Roles claim filter examples

Given an example set of groups retrieved from a hypothetical identity provider:

```
it-admin
it-developer
devops-user
devops-admin
devops-developer
product-user
product-developer
org-user
hr-user
hr-admin
```

Basic exact match filters

```
- exactMatch: "product-user"
- exactMatch: "org-user"
```

returns:

```
product-user
org-user
```

Basic regular expression (RegEx) filters

```
- regex: ".*-developer"
```

returns:

```
it-developer
devops-developer
product-developer
```

Multiple regular expression (RegEx) filters

```
- regex: ".*-developer"
- regex: "^it"    # starts with "it"
- regex: "admin$" # ends with "admin"
- regex: "\w+"    # at least one word or more
```

returns:

```
it-admin
it-developer
devops-admin
devops-developer
product-developer
hr-admin
```

Note

filters are disjunctive and so multiple filters can filter down the same values, but the resulting set will always have unique values.

Exact match and regular expression (RegEx) filters

```
- exact-match: "hr-admin"
- exact-match: "org-user"
- regex: "developer$"    # ends with "developer"
```

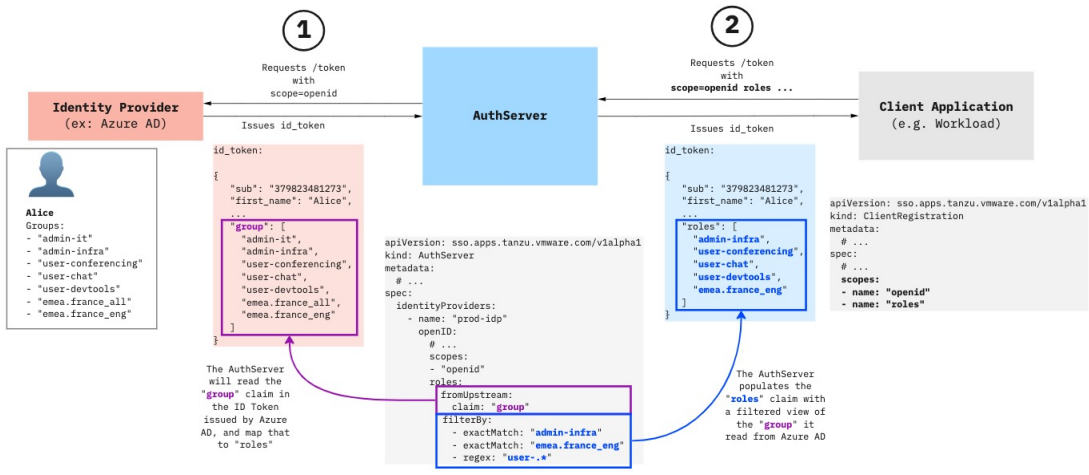
returns:

```
it-developer
devops-developer
product-developer
org-user
hr-admin
```

Roles claim mapping and filtering explained

When issuing an `id_token`, OpenID providers may include a (non-standard) claim describing the “groups” the user belongs to, the “roles” of the user, or something similar. This claim is identity provider specific. For example, Azure AD uses the “group” claim by default, and allows administrators to select the name of the claim.

Service Operators may choose to make these “groups”, “roles”, or equivalent, available in the id_token issued by an AppSSO `AuthServer`, in the `roles` claim, with filtering rules applied.



Restrictions

Each identity provider has a declared `name`. The following conditions apply:

- the names must be unique
- the names must not be blank
- the names must follow Kubernetes' DNS Subdomain Names guidelines
    - contain no more than 253 characters
    - contain only lowercase alphanumeric characters, '-' or '.'
    - start with an alphanumeric character
    - end with an alphanumeric character
- the names may not start with `client` or `unknown`

There can be at most one of each `internalUnsafe` and `ldap`.

Configure authorization for AppSSO

This topic tells you how to configure authorization for Application Single Sign-On (commonly called AppSSO).

Note

This topic is applicable to Internal, OpenID, LDAP, and SAML (experimental) identity provider `AuthServer` configurations. For more information, see AuthServer.

Overview

An application or `Workload` can protect certain resources based on user's level of authorization. Within OAuth 2, the application with protected resources, the Resource Server, verifies if the access token provided contains the scopes to perform an action on a protected resource.

The following excerpt is from a Spring Boot application, OAuth2 Resource Server, protecting its message API endpoints `/message/**`:

```
http.authorizeExchange(exchanges -> exchanges
                        .pathMatchers("/message/**").hasAuthority("SCOPE_message.rea
d")
                        .anyExchange().authenticated()
        )
```

The access token to access any endpoint under `/message/` path must have `message.read` scope within its access token.

For full example, see Spring Security documentation.

The following sections describe how to configure the mapping of user roles to authorization scopes at `AuthServer` identity provider and `ClientRegistration` levels.

Retrieving external groups or roles

Important

Skip this section if you work with an internal unsafe provider. External groups mapping is not required because roles are defined in the specifications.

To configure authorization for an identity provider, you must define from which claim or attribute the upstream identity provider supplies the groups or roles that a user is part of:

- OpenID external groups mapping
- LDAP external groups mapping
- SAML (experimental) external groups mapping

After external groups mapping is complete, and groups or roles are retrievable, you can optionally filter the roles that are appended to an identity token. For more information about how to filter roles, see Roles claim filtering.

Mapping individual roles into authorization scopes

After external groups or roles are mapped to AppSSO's `roles` claim, and optionally filtered for the desired set of retrieved roles, you can map each role to the desired authorization scopes.

For example, given a retrieved role "hr", any client authorizing by using `my-openid-provider` can request scopes `hr.read` or `hr.write`, provided that the client registered the scopes in `ClientRegistration.spec.scopes`:

```
kind: AuthServer
# ...
spec:
  identityProviders:
    - name: my-openid-provider
      openid:
        accessToken:
          scope:
            rolesToScopes:
              - fromRole: "hr" # -> Role "hr" is mapped to the "hr.read" and "hr.writ
e" scopes.
                toScopes:      #    Only users with the "hr" role can be issued access
token with these scopes.
                  - "hr.read"  # ^^
                  - "hr.write" # ^^
```

For example, given that a `ClientRegistration` is applied to include `hr.read` or `hr.write`:

```
kind: ClientRegistration
# ...
spec:
    scopes:
    - name: "roles" # Must request special 'roles' scope.
    - name: "hr.read"
    - name: "hr.write"
```

Any client can request an access token with the scopes, but an access token can be issued with those scopes only if the user that is being authorized has the role `hr` in the upstream identity provider.

If the user has the role `hr`, he or she must consent to allow the application to request the scopes. After the consent is provided, the user can access resources limited to the `hr.read` and `hr.write` scopes within the application by using their access token.

Default authorization scopes

You can define authorization scopes that are automatically granted to all users within an identity provider, regardless of user role.

For example, given an `AuthServer` with an OpenID provider, with defined authorization scope defaults:

```
kind: AuthServer
# ...
spec:
    identityProviders:
    - name: my-openid-provider
      openid:
        accessToken:
            scope:
                defaults:
                - "developer.read"
                - "developer.write"
                - "developer.delete"
```

For example, given that a `ClientRegistration` is applied to include any of the default scopes:

```
kind: ClientRegistration
# ...
spec:
    scopes:
    - name: "roles" # Must request special 'roles' scope.
    - name: "developer.read"
```

When an application or `Workload` is registered by using the `ClientRegistration`, that application, on behalf of the user, can request and be granted with the scope `developer.read` automatically within the issued access token. The user must consent to allow the application to request the scope. After the consent is provided, the user can access resources limited to the `developer.read` scope within the application by using their access token.

The following is a full sample of authorization configurations and the accompanying `ClientRegistration` configurations to allow clients to request the scopes:

```
kind: AuthServer
# ...
spec:
  identityProviders:
    - name: my-openid-provider
      openid:
```

```
        roles:
          fromUpstream:
            claim: "groups"            # -> Map the upstream identity provider's exter
nal groups or roles claim.
          filterBy:                    # -> Optionally filter the groups or roles retr
ieved from identity provider.
            - exactMatch: "finance"    # ^^
            - exactMatch: "hr"         # ^^
            - exactMatch: "marketing"  # ^^
        accessToken:
          scope:
            defaults:                  # -> Optional default scopes granted to any use
r within the identity provider.
              - "developer.read"       # ^^
              - "developer.write"      # ^^
              - "developer.delete"     # ^^
            rolesToScopes:
            - fromRole: "hr"           # -> Role "hr" is mapped to "hr.read", "hr.writ
e" scopes.
              toScopes:                #    Only users with "hr" role can be issued ac
cess token with these scopes.
                - "hr.read"            # ^^
                - "hr.write"           # ^^
            - fromRole: "finance"      # -> Role "finance" is mapped to "finance" scop
e.
              toScopes:                #    Only users with "finance" role can be issu
ed an access token with this scope.
                - "finance"            # ^^
            - fromRole: "marketing"    # -> Role "marketing" is mapped to "marketing-r
eader", "marketing-writer" scopes.
              toScopes:                #    Only users with "marketing" role can be is
sued an access token with these scopes.
                - "marketing-reader"   # ^^
                - "marketing-writer"   # ^^
```

```
kind: ClientRegistration
# ...
spec:
  scopes:
    - name: "roles" # Must request special 'roles' scope.
    - name: "developer.read"
    - name: "developer.write"
    - name: "developer.delete"
    - name: "hr.read"
    - name: "hr.write"
    - name: "finance"
    - name: "marketing-reader"
    - name: "marketing-writer"
```

Public clients and CORS for AppSSO

This topic tells you how to configure Application Single Sign-On (commonly called AppSSO) to use public clients.

Overview

A public client is a client application that does not require credentials to obtain tokens, such as single-page apps (SPAs) or mobile devices. Public clients rely on Proof Key for Code Exchange (PKCE) Authorization Code flow extension.

Follow these steps to configure an `AuthServer` and `ClientRegistration`s for use with public clients:

1. Specify allowed HTTP origin (one or more) by using the `AuthServer.spec.cors` API.

The authorization server relaxes the same-origin policy for the specified domain (one or more), enabling browser-based, single-page applications to interact with the designated authorization server. For more information, see CORS configuration.

2. Set `clientAuthenticationMethod` to `none` within `ClientRegistration` resource.

   Native applications and browser-based applications are considered public clients and must not rely on client credentials. Instead, PKCE must be used. Setting `clientAuthenticationMethod: none` ensures client credentials are not used, and makes PKCE mandatory for those clients. For more information, see Client authentication.

CORS configuration

A browser-based public client can interact with an `AuthServer` if the `AuthServer` has the public clients' origin (one or more) specified in `AuthServer.spec.cors`.

VMware recommends designating specific origins as follows:

```
kind: AuthServer
# ...
spec:
  cors:
    allowOrigins:
    - "https://example.com"         # Specific domain.
    - "https://mydept.example.com" # Specific subdomain.
    - "https://*.example.com"       # Subdomain wildcard notation.
    - "https://*.apps.example.com" # Subdomain wildcard notation.
```

You can also designate that all origins are allowed as follows:

```
kind: AuthServer
metadata:
  annotations:
    sso.apps.tanzu.vmware.com/allow-unsafe-cors: ""
spec:
  cors:
    allowAllOrigins: true
```


Caution

Do not allow all origins in production environments.

You must use the `allow-unsafe-cors` annotation when allowing all origin allowance. The `AuthServer` sends the `Access-Control-Allow-Origin: *` HTTP response header.

Requirements for allowed origin designations include:

- Only `allowOrigins` or `allowAllOrigins` is allowed to be set.
- When using `allowAllOrigins`, you must explicitly set the annotation `sso.apps.tanzu.vmware.com/allow-unsafe-cors: ""`. This is an acknowledgement that using `allowAllOrigins` is inherently unsafe.

Client authentication

When configuring a `ClientRegistration` for a public client, you must set your client authentication method to `none` and ensure that your public client supports Authorization Code with PKCE. With PKCE, the client does not authenticate, but presents a code challenge and subsequent code verifier to establish trust with the authorization server.

To set Client Authentication Method to `none`, ensure your `ClientRegistration` resource defines the following:

```
kind: ClientRegistration
# ...
spec:
  clientAuthenticationMethod: none
```

Public clients supporting Authorization Code with PKCE flow ensure that:

- On every OAuth `authorize` request, parameters `code_challenge` and `code_challenge_method` are provided. Only `code_challenge_method=S256` is supported.
- On every OAuth `token` request, parameter `code_verifier` is provided. Public clients do not provide a Client Secret because they are not tailored to retain any secrets in public view.

For public clients, the `AuthServer` only supports the Authorization Code Flow: `response_type=code`, because public clients such as single page apps cannot safely store sensitive information such as client secrets.

References

- Proof Key for Code Exchange (PKCE) specification.
- PKCE code challenge/verifier example.
- Client types - OAuth 2.1 Draft 7 specification.

Token settings for Application Single Sign-On

This topic tells you how to configure token expiry settings for Application Single Sign-On (commonly called AppSSO).

Token expiry

AppSSO allows you to optionally configure the token expiry settings in your `AuthServer` resource.

The default token expiry settings are as follows:

Token type	Lifetime
Access token	12 hours
Identity token	12 hours
Refresh token	720 hours or 30 days

VMware recommends setting a shorter lifetime for access tokens, typically measured in hours, and a longer lifetime for refresh tokens, typically measured in days. Refresh tokens acquire new access tokens, so they have a longer lifespan.

To override the token expiry settings, configure the following in your `AuthServer` resource:

```
kind: AuthServer
# ...
spec:
  token:
    accessToken:
      expiry: "12h"
    idToken:
      expiry: "12h"
```

```
    refreshToken:
      expiry: "720h"
```

`expiry` field examples:

Type	Example	Definition
Seconds	`10s`	10 seconds
Minutes	`10m`	10 minutes
Hours	`10h`	10 hours

Note

The `expiry` field adheres to the duration constraints of the Go standard time library and does not support durations in units beyond hours, such as days or weeks. For more information, see the Go documentation.

Constraints

The token expiry constraints are as follows:

- The duration of the `expiry` field cannot be negative or zero.
- The refresh token's expiration time cannot be the same as or shorter than that of the access token.

Verify token settings

After you set up an Application Single Sign-On `AuthServer`, you can verify that the token received by applications looks as expected. For this purpose, you can create a simple application consuming your `AuthServer`. The following YAML file creates such an application. When you access its URL, it enables you to log in by using your `AuthServer` and displays the token it receives.

Caution

- The simple application is not intended for production use. It only serves a tool to help you verify your setup.
- The following YAML file pulls an unvetted public image `bitnami/oauth2-proxy:7.3.0`
- This section does not apply to an air-gapped environment.

If you stored the following YAML in a file named `token-viewer.yaml`, you can apply it to your cluster by running the following command:

```
  ytt -f token-viewer.yaml --data-value ingress_domain=YOUR-INGRESS-DOMAIN --data-valu
e-yaml 'authserver_selector=YOUR-AUTHSERVER-SELECTOR' | kubectl apply -f-
```

Where `YOUR-AUTHSERVER-SELECTOR` is the label name and its value. For example: `{"name": "ci"}`.

A full example is as follows:

```
#!
#! Token viewer
#!
#! usage:
```

```
#!
#! ytt -f token-viewer.yaml --data-value ingress_domain=example.com --data-value-yaml
'authserver_selector={"name": "ci"}'
#!
#! Then navigate to http://token-viewer.<INGRESS_DOMAIN>
#!

#@ load("@ytt:data", "data")
#@ fqdn = "token-viewer." + data.values.ingress_domain
#@ redirect_uri = "http://" + fqdn + "/oauth2/callback"
#@ namespace = data.values.namespace if "namespace" in data.values else "default"

---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client-registration
  namespace: #@ namespace
spec:
  authServerSelector:
    matchLabels: #@ data.values.authserver_selector
  redirectURIs:
    - #@ redirect_uri
  requireUserConsent: false
  clientAuthenticationMethod: client_secret_basic
  authorizationGrantTypes:
    - "authorization_code"
  scopes:
    - name: "openid"
    - name: "email"
    - name: "profile"
    - name: "roles"

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: token-viewer
  namespace: #@ namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      name: token-viewer
  template:
    metadata:
      labels:
        name: token-viewer
    spec:
      securityContext:
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
      containers:
        - image: bitnami/oauth2-proxy:7.3.0
          name: proxy
          securityContext:
            runAsNonRoot: true
            seccompProfile:
              type: RuntimeDefault
            allowPrivilegeEscalation: false
            capabilities:
              drop:
                - ALL
          ports:
            - containerPort: 4180
```

```
            name: proxy-port
            protocol: TCP
          env:
            - name: ISSUER_URI
              valueFrom:
                secretKeyRef:
                  name: my-client-registration
                  key: issuer-uri
            - name: CLIENT_ID
              valueFrom:
                secretKeyRef:
                  name: my-client-registration
                  key: client-id
            - name: CLIENT_SECRET
              valueFrom:
                secretKeyRef:
                  name: my-client-registration
                  key: client-secret
          command: [ "oauth2-proxy" ]
          args:
            - --oidc-issuer-url=$(ISSUER_URI)
            - --client-id=$(CLIENT_ID)
            - --insecure-oidc-skip-issuer-verification=true
            - --client-secret=$(CLIENT_SECRET)
            - --cookie-secret=0000000000000000
            - --cookie-secure=false
            - --http-address=http://:4180
            - --provider=oidc
            - --scope=openid email profile roles
            - --email-domain=*
            - --insecure-oidc-allow-unverified-email=true
            - --oidc-groups-claim=roles
            - --upstream=http://127.0.0.1:8000
            - #@ "--redirect-url=" + redirect_uri
            - --ssl-upstream-insecure-skip-verify=true
            - --ssl-insecure-skip-verify=true
            - --skip-provider-button=true
            - --pass-authorization-header=true
            - --prefer-email-to-user=true
        - image: python:3.9
          name: application
          securityContext:
            runAsNonRoot: true
            runAsUser: 1001
            seccompProfile:
              type: RuntimeDefault
            allowPrivilegeEscalation: false
            capabilities:
              drop:
                - ALL
          resources:
            limits:
              cpu: 100m
              memory: 100Mi
          command: [ "python" ]
          args:
            - -c
            - |
              from http.server import HTTPServer, BaseHTTPRequestHandler
              import base64
              import json

              class Handler(BaseHTTPRequestHandler):
                  def do_GET(self):
                      if self.path == "/token":
                          self.token()
```

```
                        elif self.path == "/jwt":
                            self.jwt()
                        else:
                            self.greet()

                    def greet(self):
                        username = self.headers.get("x-forwarded-user")
                        self.send_response(200)
                        self.send_header("Content-type", "text/html")
                        self.end_headers()
                        page = f"""
                        <h1>It Works!</h1>
                        <p>You are logged in as <b>{username}</b></p>
                        <p><a href="/jwt">Show me my id_token (JWT format)</a></p>
                        <p><a href="/token">Show me my id_token (JSON format)</a></p>
                        """
                        self.wfile.write(page.encode("utf-8"))

                    def token(self):
                        token = self.headers.get("Authorization").split("Bearer ")[-1]
                        payload = token.split(".")[1].replace("-","+").replace("_","/")
                        decoded = base64.b64decode(bytes(payload, "utf-8") + b'==').deco
de("utf-8")
                        self.send_response(200)
                        self.send_header("Content-type", "application/json")
                        self.end_headers()
                        self.wfile.write(decoded.encode("utf-8"))

                    def jwt(self):
                        token = self.headers.get("Authorization").split("Bearer ")[-1]
                        self.send_response(200)
                        self.send_header("Content-type", "text/plain")
                        self.end_headers()
                        self.wfile.write(token.encode("utf-8"))

                server_address = ('', 8000)
                httpd = HTTPServer(server_address, Handler)
                httpd.serve_forever()

---
apiVersion: v1
kind: Service
metadata:
  name: token-viewer
  namespace: #@ namespace
spec:
  ports:
    - port: 80
      targetPort: proxy-port
      name: proxy-svc-port
  selector:
    name: token-viewer

---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: token-viewer
  namespace: #@ namespace
spec:
  rules:
    - host: #@ fqdn
      http:
        paths:
          - path: "/"
            pathType: Prefix
```

```
            backend:
              service:
                name: token-viewer
                port:
                  name: proxy-svc-port
```

Token signatures for AppSSO

This topic tells you how to configure token signatures keys for Application Single Sign-On (commonly called AppSSO).

Overview

An `AuthServer` must have token signature keys configured to be able to mint tokens.

Learn about token signatures and how to manage keys of an `AuthServer`:

- Token signature 101
- Token signature in AppSSO
- Creating keys
- Rotating keys
- Revoking keys

“Token signature key” or just “key” is AppSSO’s wording for a public/private key pair that is tasked with signing and verifying JSON Web Tokens (JWTs). For more information, please refer to the following resources:

- JSON Web Signature (JWS) spec
- JSON Web Algorithms (JWA) spec
- JSON Web Token (JWT) spec

Token signature 101

Token signature keys are used by an `AuthServer` to sign JSON Web Tokens (JWTs), produce a JWS Signature and attach it to the JOSE Header of a JWT. The client application can then verify the JWT signature.

A private key signs a JWT. A public key verifies the signature of a signed JWT.

The sign-and-verify mechanism serves multiple security purposes:

- **Authenticity**: signature verification ensures that the issuer of the JWT is from a source that is advertised.
- **Integrity**: signature verification ensures that the JWT has not been altered in transit or during its issued lifetime. Integrity is a foundational pillar of the CIA triad concept in Information Security.
- **Non-repudiation**: signature verification ensures that the authorization server that signed the JWT cannot deny that they have signed it after its issuance (granted that the signing key that signed the JWT is available).

AppSSO only supports the `RS256` algorithm for signing tokens. For more information, see JSON Web Algorithms (JWA) documentation.

Token signature of an `AuthServer`

You must configure token signatures for `AuthServer`. An `AuthServer` receives its keys under `spec.tokenSignature`. For example:

```
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: sample-token-signing-key
    extraVerifyKeyRefs:
      - name: sample-token-verification-key-1
      - name: sample-token-verification-key-2
```

There can only be **one** token signing key `spec.tokenSignature.signAngVerifyKeyRef` at any given time, and arbitrarily many token verification keys `spec.tokenSignature.extraVerifyKeyRefs`. The token signing key is used to sign and verify actively issued JWTs in circulation, whereas token verification keys are used to verify issued JWTs signatures. Token verification keys are thought to be previous token signing keys but have been rotated into verify only mode as a rotation mechanism measure, and can potentially be slated for eviction at a predetermined time.

The `AuthServer` serves its public keys at `{spec.issuerURI}/oauth2/jwks`. For example:

```
❯ curl -s authserver-sample.default/oauth2/jwks | jq
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "sample-token-signing-key",
      "n": "0iCinir7sWKZE_3QXq4eTub_GU-lvdAKFI9dzDlwX7XZwwSERuzzQQ_Fs7i9djMl5bpv2ma_3Z
B-j2W9pR9ZIa3nqBI29AHqx2zmVQ8w-GxPDGRMkBdMOWNwyDQGIRlQnJFpXRoSQ5_viM9gYA56WthkDghrupGU
iB_zqGFYlgnz7sd4lC-thgEkDi9vY68DLIFdsXOQIXFqakyEIo43n_0vg6JRGQW1LU_32Ok6OgA3r6bYcE8VQh
JW3sE1qOSFcP0JrPA3YgmTNuDV6GoCLZeMxDdMDKdDcH5UgERLQe1qMMKwlMCeKamOWgo9eBvcFnWNR0I_MJV6
F14U1WbIcQ"
    },
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "sample-token-verification-key-1",
      "n": "wc7uOACU62Yu_zKT9YrI4v-_X3L47nbVlcByi4UTVhg8o001OkiYAPAEoDCEHnDg_54gTWxe3h
DRcOJrd72PkTAaxH8aFdikoyakRVG9NvAPbcfzvI8R8plepUbs1U7TPPDEDARm_fZX6QdVyz0CTSafrz-yktTA
DxJhYPgvFLeHq7g7RouB1szTWDCM1haoxKa4960_x9meghNn87z0uF3cAd7TM_k3capYnxNOUT5g1vjJ05Vk14
JUl4R294OpMXPCGcFuvu9auXeBqXyKxxTAnLkDdNrgtT0FJHwnh4RGnrNqjYZOwlRvGbzwQ7du97aU2-qgbKkJ
rWYZWcw2bQ"
    },
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "sample-token-verification-key-2",
      "n": "qELrLiaD-IVp_nthVn2EsLuShtU9ovyVIPkLVf47AqKogPV2frE_6Sv8k7Zim-SgDXfjLEg-UG
lQrb4KFm_WkaK2Uf6PCapiBnMi1Q5P8qC0WC5LT6XyPY1exCQbMrEsyd89oS0sKxgoc3Qv0XV24jGYiWQyJ7I0
Rub_QEldGM_dSlfbI-1Qt_U6Ll22OEc1D6P1A3MdDrgbur6N7ZemxlKI26-OAdlbNi0u-lFNj3Ss-pfTVi_fD2
hAajRRmc4tmHejQjH36M4F1NSW_gTbb6VX5EerVuDwSCCK0EuGvhcb1hg6kYEoO-qws54AQ0PywBXT5qksCMBm
mzjP6qO4Ow"
    }
  ]
}
```

Caution Changes to `spec.tokenSignature.signAngVerifyKeyRef` have immediate effects.

As a *service operator*, you have control over which keys are used for certain purposes. Navigate to the next few sections for more information.

Creating keys

You can deploy an `AuthServer` without `spec.tokenSignature` but it won't be able to mint tokens. Therefore, keys must be configured to make it fully operational. The following describe how to create and apply a keys for an `AuthServer`.

An RSA key can be created multiple ways. Below are two recommended approaches – choose one.

Using secretgen-controller

Important

This section assumes you have Tanzu Application Platform running on your cluster with `secretgen-controller` installed.

An RSAKey CR allows for expedited creation of a Secret resource containing PEM-encoded public and private keys required by an `AuthServer`.

1. Create an `AuthServer` with `RSAKeys` as follows:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
 name: authserver-sample
 namespace: default
spec:
 tokenSignature:
   signAndVerifyKeyRef:
     name: my-token-signing-key
   extraVerifyKeyRefs:
     - name: my-token-verification-key
 # ...
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
 name: my-token-signing-key
 namespace: default
spec:
 secretTemplate:
   type: Opaque
   stringData:
     key.pem: $(privateKey)
     pub.pem: $(publicKey)
---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
 name: my-token-verification-key
 namespace: default
spec:
 secretTemplate:
   type: Opaque
   stringData:
     key.pem: $(privateKey)
     pub.pem: $(publicKey)
```

2. Observe the creation of an underlying `Secrets`. The name of the each `Secret` is the same as the `RSAKey` names:

```
# Verify Secret exists
kubectl get secret my-token-signing-key
```

```
# View the base64-encoded keys
kubectl get secret my-token-signing-key -o jsonpath='{.data}'
```

You should be able to see two fields within the Secret resource: `key.pem` (private key) and `pub.pem` (public key).

3. Verify that the `AuthServer` serves its keys

```
curl -s authserver-sample.default/oauth2/jwks | jq
```

If you encounter any issues with this approach, see Carvel Secretgen Controller documentation.

Using OpenSSL

You can generate an RSA key yourself using OpenSSL. Here are the steps:

1. Generate a PEM-encoded RSA key pair

   This guide references the freely published OpenSSL Cookbook and the approaches mentioned therein around generating a public and private key pair.

```
# Generate an 4096-bit RSA key
openssl genpkey -out privatekey.pem -algorithm RSA -pkeyopt rsa_keygen_bits:409
6
# -> privatekey.pem
# The resulting private key output is in the PKCS#8 format

# Next, extract the public key
openssl pkey -in privatekey.pem -pubout -out publickey.pem
# -> publickey.pem
# The resulting public key output is in the PKCS#8 format

# To view details of the private key
openssl pkey -in privatekey.pem -text -noout
```

   For OpenSSL key generation examples, see the OpenSSL documentation.

Important

The minimum key size for an `Authserver` is 2048.

2. Create a secret resource by using the key generated earlier in this procedure:

```
kubectl create secret generic my-key \
--from-file=key.pem=privatekey.pem \
--from-file=pub.pem=publickey.pem \
--namespace default
```

3. Apply your `AuthServer`:

```
  apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
  kind: AuthServer
  metadata:
    name: authserver-sample
    namespace: default
  spec:
    tokenSignature:
      signAndVerifyKeyRef:
        name: my-key
    # ...
```

4. Verify that the `AuthServer` serves its keys

```
curl -s authserver-sample.default/oauth2/jwks | jq
```

Rotating keys

This section describes how to “rotate” token signature keys for an `AuthServer`.

The action of “rotating” means moving the active token signing key into the set of token verification keys, generating a new cryptographic key, and assigning it to be the designated token signing key.

Assuming that you have an `AuthServer` with token signature keys configured, rotate keys as follows:

1. Generate a new token signing key first. See creating keys. Verify that the new `Secret` exists before proceeding to the next step.
2. Edit `AuthServer.spec.tokenSignature`, append the existing `spec.tokenSignature.signAndVerifyKeyRef` to `spec.tokenSignature.extraVerifyKeys` and set your new key as `spec.tokenSignature.signAndVerifyKeyRef`.

   For example:

```
# Before
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
 name: authserver-sample
 namespace: default
spec:
 tokenSignature:
   signAndVerifyKeyRef:
     name: old-key
   extraVerifyKeys: []
 # ...
```

```
# After
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
 name: authserver-sample
 namespace: default
spec:
 tokenSignature:
   signAndVerifyKeyRef:
     name: new-key
   extraVerifyKeys:
     - name: old-key
 # ...
```

   Once you apply your changes, key rotation is effective immediately.

Moving the active token signing key to be a token verification key is an *optional* step – check out the Revoking keys section for more.

Revoking keys

This section describes how to “revoke” token signature keys for an `AuthServer`.

The action of “revoking” a key means to entirely remove the key from circulation by an `AuthServer`, whether it be a token signing key or a token verification key. This action might be needed if your organization requires a complete key refresh where older keys are never retained. Another scenario might be in the case of an emergency in which a key or a session has been compromised and a complete revocation is warranted.

To revoke an existing key or keys, you may remove any references to the keys in the `spec.tokenSignature` resource. By removing the reference to the key, the system shall no longer acknowledge that the key is used for signing or verifying JWTs.

For example, if you have a token signing key and a few verification keys:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: authserver-sample
  namespace: default
spec:
  tokenSignature:
    signAndVerifyKeyRef:
      name: key-3
    extraVerifyKeys:
      - name: key-2
      - name: key-1
  # ...
```

To revoke an existing verification key, remove it from the `extraVerifyKeys` list. In the example above, you can remove “key-2” and “key-1” from the list; JWTs signed with those keys will no longer be verifiable.

To revoke an existing token signing key, remove it from `signAndVerifyKeyRef` field. However, if you remove an existing token signing key without a replacement key, the `AuthServer` will not be able to issue access tokens until a valid token signing key is provided. In the example above, “key-3” would be removed; the system will not be able to sign or verify JWTs.

References and further reading

- JSON Web Signature (JWS) - rfc7515 (ietf.org)
- JSON Web Algorithms (JWA) spec
- JSON Web Token (JWT) - rfc7519 (ietf.org)

Storage for AppSSO

This topic tells you how to configure the storage for Application Single Sign-On (commonly called AppSSO).

Overview

AppSSOs `AuthServer` handles data pertaining to user’s session, identity, access tokens and approved or rejected consents. For production environments, it is critical to provide your own storage source to enable enterprise functions such as data backup and recovery, auditing and long-term persistence according to your organization’s data and security policies.

AppSSO currently only supports Redis `v6.0` or above as a storage solution. `v6.0` introduced TLS support to ensure encrypted client-server communication - AppSSO enforces TLS by default.

Storage provided by default refers to an `AuthServer` resource where `.spec.storage` is not set.

Although data in motion is encrypted by using TLS, data at rest is not encrypted by default through `AuthServer`. Each storage provider is responsible for encrypting their own data. See data types for more information about storage.

Securing Data at rest

To be compliant with HIPAA, FISMA, PCI and GDPR, you must encrypt data at rest. Securing the underlying infrastructure that Redis uses is crucial to protect against a potential attack. The National Institute for Standards and Technology – Federal Information Processing Standards (NIST-FIPS) sets the standard for best practice when it comes to data security in the US. Symmetric cryptography can be used to protect data at rest. This means that the same key encrypts and decrypts the data, so there is no need for a different private and public key. The Advanced Encryption Standard (AES) encryption algorithm is an industry standard for securing data at rest. For the highest level security, VMware recommends using a 256-bit key.

Configuring Redis

To configure Redis as authorization server storage, you must have the following information of your Redis server:

- **Server CA certificate** (optional): the Certificate Authority (CA) certificate to verify Redis TLS connections. It is not required if Redis Server certificate is signed by a public CA.
- **host** (required): the domain name, IP address, or host name of your Redis server.
- **port** (optional): the port number of your Redis server. It default to `6379` and must be a string.
- **username** (optional): the username to authenticate against your Redis server.
- **password** (optional): the password to authenticate against your Redis server.

AppSSO takes the secure-by-default approach and does not establish non-encrypted communication channels. The `AuthServer` resource enters an error state if a non-encrypted connection is attempted.

mTLS is not supported, however Vanilla Redis uses mTLS by default. It can be turned off by setting `tls-auth-clients no`. For more information, see Redis documentation.

The following steps introduce the path to configuring Redis with AppSSO:

1. Configuring Redis Server CA certificate
2. Configuring a Redis Secret
3. Attaching storage to an AuthServer

Configuring Redis Server CA certificate

If your Redis includes a custom or non-public Server CA certificate, you must instruct AppSSO to trust the CA certificate. This is required for the authorization server to communicate with your Redis over TLS. See CA certificates for more information about configuring a CA certificate with AppSSO.

Configuring a Redis Secret

To provide coordinates (the location details) of your Redis server, you must create a `Secret` resource that follows well-known Secret entries conventions. For more information, see Service Bindings 1.0.0 specification.

Example of a properly formatted `Secret` resource:

Important

The Secret must be created in the same namespace as your `AuthServer`.

```
apiVersion: v1
kind: Secret
metadata:
  name: redis-credentials
  namespace: my-authserver
type: servicebinding.io/redis        # optional, must equal 'servicebinding.io/redis'
if defined
stringData:
  type: "redis"                      # required, must equal 'redis'
  ssl: "true"                        # required, must equal 'true'
  host: "redis01.prod.example.com"   # required
  port: "6379"                       # optional, must be a string, defaults to "6379"
if left empty
  password: "!!veryStrongPassword!!" # optional
  username: "redis01-user"           # optional
```

Attaching storage to an AuthServer

After a Redis Secret resource is applied, you can reference the Secret in `.spec.storage`. An example of an AuthServer with a reference to a Redis Secret is as follows:

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: my-authserver-example
  namespace: my-authserver
spec:
  # ...
  storage:
    redis:
      serviceRef:
        apiVersion: "v1"
        kind: "Secret"
        name: redis-credentials
```

After `AuthServer` is applied, ensure its `Status` is `Ready`.

Inspecting storage of an AuthServer

You can inspect the status of an `AuthServer`'s storage as follows:

```
kubectl get authserver <authserver-name> \
  --namespace <authserver-namespace> \
  --output jsonpath="{.status.storage.redis}" | jq
```

Expect to see the following output with the actual Redis host and port:

```
{
  "redis": {
    "host": "ci-redis.authservers.svc.cluster.local",
    "port": "6379"
  }
}
```

Storage provided by default

If no storage is defined, an `AuthServer` provides its own short-lived ephemeral storage solution, Redis. The provided Redis is configured to never flush any data to any volume that might be attached to the pods that operate the authorization server.

Caution

The default storage configuration is desisged for prototyping or testing environments and must not be used in production environments.

To view details for Redis of an `AuthServer`:

```
# Get the Redis image
kubectl get authserver <authserver-name> \
  --namespace <authserver-namespace> \
  --output jsonpath="{.status.deployments.redis}" | jq

# Get the Redis host and port
kubectl get authserver <authserver-name> \
  --namespace <authserver-namespace> \
  --output jsonpath="{.status.storage.redis}" | jq
```

Data types

The following data is stored in Redis:

- Client information
  - Authorization grant type
  - Client id
- User session
  - Session token
  - Refresh token
- Identity and access tokens

  Note

  This is the data that carries the highest level risk.

  - Authentication token including the principal
    - Personally identifying information such as email and name
- Approved or rejected consents
  - A client identifier
  - A reference to the user
  - A list of the Authorities that the user has granted to this client

Known limitations of storage providers

Redis Cluster

When your storage is provided by Redis Cluster, additional settings might be required.

The nodes and the maximum number of redirects must be set in your Service Bindings' `Secret`. For example, in addition to the entries in Configuring a Redis Secret, you must provide `cluster` settings as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: redis-cluster-credentials
  namespace: authservers
type: servicebinding.io/redis
stringData:
  #...
  cluster.max-redirects: 5
  cluster.nodes: 100.90.1.10:6379,100.90.1.11:6379,100.90.1.12:6379
```

Important

`cluster.nodes` must be a comma-separated list of `<ip>:<port>`.

AuthServer readiness for AppSSO

This topic tells you how to use `AuthServer.status` as a reliable source to verify an `AuthServer`'s readiness for Application Single Sign-On (commonly called AppSSO).

You can verify your `AuthServer` by ensuring:

- there is at least one token signing key configured.

  ```
  curl -X GET {spec.issuerURI}/oauth2/jwks
  ```

  The response body should yield at least one key in the list. If there are no keys, please apply a token signing key

- OpenID discovery endpoint is available.

  ```
  curl -X GET {spec.issuerURI}/.well-known/openid-configuration
  ```

  The response body should yield a valid JSON body containing information about the `AuthServer`.

Client registration check

It is helpful to verify an `AuthServer` by running a test run with a test `ClientRegistration`. It ensures that app developers can register clients with the `AuthServer` successfully.

Follow the steps below to ensure that your installation can:

1. Add a test client.
2. Get an access token.
3. Invalidate/remove the test client.

Prerequisites

Ensure that you have successfully applied a token signing key to your `AuthServer` before proceeding.

Define and apply a test client

Apply a `ClientRegistration` to your cluster in a Namespace that the `AuthServer` should allow clients from:

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: test-client
  namespace: default
spec:
  authServerSelector:
    matchLabels:
    # appropriate labels for your `AuthServer`
  authorizationGrantTypes:
    - client_credentials
  clientAuthenticationMethod: client_secret_basic
```

See the ClientRegistration API reference for more field definitions.

This defines a test `ClientRegistration` with the `client_credentials` OAuth grant type.

Apply the `ClientRegistration`:

```
kubectl apply -f appsso-test-client.yaml
```

Once the `ClientRegistration` is applied, inspects its status and verify it's ready.

Get an access token

You should be able to get a token with the client credentials grant for example:

```
# Get client id (`base64` command has to be available on the command line)
export APPSSO_TEST_CLIENT_ID=$(kubectl get secret test-client -n default -o jsonpath="
{.data['client-id']}" | base64 --decode)

# Get client secret (`base64` command has to be available on the command line)
export APPSSO_TEST_CLIENT_SECRET=$(kubectl get secret test-client -n default -o jsonpa
th="{.data['client-secret']}" | base64 --decode)

# Attempt to fetch access token
curl \
 --request POST \
 --location "{spec.issuerURI}/oauth2/token" \
 --header "Content-Type: application/x-www-form-urlencoded" \
 --header "Accept: application/json" \
 --data "grant_type=client_credentials" \
 --basic \
 --user $APPSSO_TEST_CLIENT_ID:$APPSSO_TEST_CLIENT_SECRET
```

You should see a response JSON containing populated field `access_token`. If so, the system is working as expected, and client registration check is successful.

Make sure to delete the test `ClientRegistration` once you are done.

Scale AuthServer for AppSSO

This topic tells you how to scale `AuthServer` for Application Single Sign-On (commonly called AppSSO).

The number of authorization server replicas for an `AuthServer` can be specified under `spec.replicas`.

Furthermore, `AuthServer` implements the `scale` subresource. That means you can scale an `AuthServer` with the existing tooling. For example:

```
kubectl scale authserver authserver-sample --replicas=3
```

The resource of the authorization server and Redis `Deployments` can be configured under `spec.resources` and `spec.redisResources` respectively. See the API reference for details.

AuthServer audit logs for AppSSO

This topic tells you how to use `AuthServer` audit logs in Application Single Sign-On (commonly called AppSSO).

Overview

`AuthServer`s perform the following tasks:

- Handle user authentication
- Issue `id_token` and `access_token`

Each audit event contains the following information:

- `ts` - date/time of the event
- `remoteIpAddress` - the IP of the user-authentication or if not attainable, the IP of the last proxy

Authentication

`AuthServer` produce the following authentication events:

- `AUTHENTICATION_SUCCESS`
  - **Trigger** successful authentication
  - **Data recorded** Username, Provider ID, Provider Type (INTERNAL, OPENID, ...)
- `AUTHENTICATION_LOGOUT`
  - **Trigger** successful logout
  - **Data recorded** Username, Provider ID, Provider Type (INTERNAL, OPENID, ...)
- `AUTHENTICATION_FAILURE`
  - **Trigger** failed authentication using either `internalUnsafe` or `ldap` identity provider
  - **Data recorded** Username, Provider ID, Provider Type (INTERNAL or LDAP)
- `INVALID_IDENTITY_PROVIDER_CONFIGURATION`
  - **Trigger** some cases of failed authentication with an `openId` or `saml` identity provider
  - **Data recorded** Provider ID, Provider Type, error
  - **Note** usually followed by a human-readable help message, with `"logger": "appsso.help"`

Token flows

`AuthServer` produce the following authorization_code and token events:

- `AUTHORIZATION_CODE_ISSUED`
  - **Trigger** `authorization_code` grant type, successful call to `/oauth2/authorize`

    - **Data recorded** Username, Provider ID, Provider Type, Client ID, Scopes requested, Redirect URI

- `AUTHORIZATION_CODE_REQUEST_REJECTED`
    - **Trigger** `authorization_code` grant type, unsuccessful call to `/oauth2/authorize`, for example invalid Client ID, invalid Redirect URI, ...
    - **Data recorded** Error, Error Code (ex: `invalid_scope`), Client ID, Scopes requested Redirect URI, Username (may be `anonymousUser`), Provider ID and Provider Type if available
- `TOKEN_ISSUED`
    - **Trigger** successful call to `/oauth2/token`
    - **Data recorded** Scopes, Client ID, Grant Type (`authorization_code` or `client_credentials`), Username
- `TOKEN_REQUEST_REJECTED`
    - **Trigger** unsuccessful call to `/oauth2/token`, for example invalid Client Secret
    - **Data recorded** Client ID, Scopes requested, Error

Application Single Sign-On for App Operators

The following topics tell you how to secure a sample app with Application Single Sign-On (commonly called AppSSO):

- Configure AppSSO for workloads
- Secure a Spring Boot workload
- Secure a single-page app workload

Application Single Sign-On for App Operators

The following topics tell you how to secure a sample app with Application Single Sign-On (commonly called AppSSO):

- Configure AppSSO for workloads
- Secure a Spring Boot workload
- Secure a single-page app workload

Configure AppSSO for workloads

This topic tells you how to configure Application Single Sign-On (commonly called AppSSO) when defining workloads.

For specific stack implementations such as Spring Boot, see Application Single Sign-On for App Operators.

An AppSSO `AuthServer` and your `Workload` must be able to detect each other and can communicate bidirectionally:

- To make `AuthServer` detect your `Workload`, for example, `AuthServer` is responsible for authentication and authorization duties, you must create and apply a `ClientRegistration` resource. For more information, see The ClientRegistration resource.
- To make your `Workload` detect an `AuthServer`, for example, your application relies on AppSSO for authentication and authorization requests, you must specify a service resource

claim of a `ClientRegistration`, which produces the necessary credentials for your `Workload` to consume. For more information, see Claim a ClientRegistration.

- (Optional) Ensure your `Workload` trusts a TLS-enabled `AuthServer`. For more information, see Configure Workloads to trust a custom Certificate Authority (CA).

Important

You must ensure `Workload` trusts the `AuthServer` if you use the default self-signed certificate `ClusterIssuer` while installing Tanzu Application Platform.

The following sections elaborate on these concepts in detail.

The `ClientRegistration` resource

A `ClientRegistration` registers a client entity with an `AuthServer`.

Example: A `ClientRegistration` resource, residing in a workloads-ready namespace `my-apps`.

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-workload-client
  namespace: my-apps
spec:
  authServerSelector:
    matchLabels:
    # Ask your Service Operator for labels to target an `AuthServer`
    # or run `kubectl label --list authserver/<authserver-name> -n <authserver-ns>`
  authorizationGrantTypes:
    - authorization_code
  clientAuthenticationMethod: client_secret_basic
  requireUserConsent: true
  redirectURIs:
    - "<MY_WORKLOAD_HOSTNAME>/redirect-uri"
  scopes:
    - name: openid
```

To verify the status of the `ClientRegistration`, run:

```
kubectl get clientregistration my-workload-client --namespace my-apps
```

After a `ClientRegistration` is applied and has status of `Ready`, it is claimable by a `Workload`, which provides the necessary credentials to your application. For more information, see Claim a ClientRegistration.

For more information about the schema and specification of the resource, see ClientRegistration.

Redirect URIs

A client registration's `redirectURIs` define the location for an `AuthServer` to send the user back to after they are authenticated. You must configure the redirect locations based on your implementation method.

Caution

> Redirect URIs might not contain loopback alias `localhost`, for example, `http://localhost:8080`. You can use `127.0.0.1` instead, for example, `http://127.0.0.1:8080`.

For servlet-based Spring Boot workloads using Spring Security OAuth 2 Client library, the default redirect URI template is: `{workloadBaseUrl}/login/oauth2/code/{ClientRegistration.metadata.name}`. For more information about the format, see Spring documentation.

Example: If a `Workload` base domain is `https://app.my-apps.prod.example.com`, where `prod.example.com` is your `shared.ingress_domain` value and `ClientRegistration` is named as `my-workload-client`, the redirect URI is:

```
spec:
  redirectUris:
    - "https://app.my-apps.prod.example.com/login/oauth2/code/my-workload-client"
```

Authorization grant types

AppSSO supports the following OAuth grant types:

- Authorization Code: `authorization_code`

  This grant type is used by applications seeking to authenticate and authorize end-users. An `AuthServer` issues identity and access tokens to applications to identify end users' identity and the level of access they have to protected resources.

- Client Credentials: `client_credentials`

  This grant type is used by applications seeking to communicate directly to other protected applications, by using a client identifier and client secret, for example, service-to-service communication. An `AuthServer` issues access tokens that define the level of access that the requesting service has to the protected service they seek to communicate with.

  > **Note**
  >
  > Use cases for grant types `authorization_code` and `client_credentials` are typically different, so VMware recommends creating separate client registrations for each grant type.

- Refresh Token: `refresh_token`

  This grant type is used by applications seeking to obtain access tokens. If `refresh_token` grant type is included, on every access token issue by an `AuthServer`, a refresh token is included. You can use the refresh token to fetch new access tokens before older ones expire to continue accessing protected resources.

Client authentication method

Client authentication method is the approach the client takes to authenticate with an authorization server. The default value of `client_secret_basic` is the recommended method for authenticating server-based applications such as Spring Boot or .NET Core apps (confidential clients).

For browser-based single-page apps, client authentication method must be set to `none`. For more information, see Configure authorization and Client authentication methods.

Scopes

The `scopes` field allows for configuring requested OAuth2 scopes including standard OpenID claims. The scopes provided within this field can be mapped to upstream identity provider roles. For more information, see Configure authorization.

To activate issuance of uers' identity tokens and authentication, you must include the `openid` scope.

To activate fetching of user roles or groups, you must include the `roles` scope.

Example: a `ClientRegistration` allows the client to request identity tokens with user information and specific read, write, and delete developer privileges, given a user has any of the scopes listed.

```
kind: ClientRegistration
# ...
spec:
  scopes:
    - name: openid             # standard OpenID scope, containing claims "sub" (subje
ct), "aud" (audience), etc.
    - name: email              # standard OpenID scope, containing claims "email" and
"email_verified"
    - name: profile            # standard OpenID scope, containing claims "name", "giv
en_name", "family_name", etc.
    - name: roles              # AppSSO special scope, requesting user roles/groups be
populated in "roles" claim.
    - name: developer.read     # custom authorization scope
    - name: developer.write    # ^^
    - name: developer.delete   # ^^
```

Requiring user consent

The `requireUserConsent` field allows for toggling scopes approval for end-users. If activated, every end user is prompted to consent to or reject scopes that the client requests on behalf of the user. If deactivated, all scopes that the client requests are auto-approved or consented to without prompt.

Claim a `ClientRegistration`

When a `ClientRegistration` resource is ready, you can claim it by using a `ResourceClaim` resource:

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaim
metadata:
  name: my-client-claim
  namespace: my-apps
spec:
  ref:
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    kind: ClientRegistration
    name: my-workload-client
    namespace: my-apps
```

Alternatively, you can also claim a `ClientRegistration` by using tanzu service CLI:

```
tanzu service resource-claim create my-client-claim \
  --namespace my-apps \
  --resource-api-version sso.apps.tanzu.vmware.com/v1alpha1 \
  --resource-kind ClientRegistration \
  --resource-name my-workload-client \
  --resource-namespace my-apps
```

Verify the status of the service resource claim by running `tanzu service resource-claim list -n my-apps -o wide`:

```
NAMESPACE   NAME             READY  REASON  CLAIM REF
my-apps     my-client-claim  True           services.apps.tanzu.vmware.com/v1alpha1:Re
sourceClaim:my-client-claim
```

Connecting a `Workload` to an `AuthServer`

Now you can reference the created service resource claim by a `Workload`.

Example: An example `Workload` in `my-apps` namespace.

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    apps.tanzu.vmware.com/workload-type: web
  name: my-workload
  namespace: my-apps
spec:
  source:
    git:
      url: https://github.com/my-company/my-app.git
      ref:
        branch: main
  serviceClaims:
    - name: my-workload-client # Must match the name of the referenced `ClientRegistra
tion`.
      ref:
        apiVersion: services.apps.tanzu.vmware.com/v1alpha1
        kind: ResourceClaim
        name: my-client-claim
```

Alternatively, you can refer to your `ClientRegistration` when deploying your workload with the `tanzu` CLI:

```
tanzu apps workload create my-workload \
  --service-ref "my-workload-client=services.apps.tanzu.vmware.com/v1alpha1:ResourceCl
aim:my-client-claim" \
  # ...
```

Important

The service ref name must match the name of the referenced `ClientRegistration`.

The service resource claim reference binding ensures that your `Workload`'s `Pod` (one or more) is mounted with a volume containing the necessary credentials required by your application to detect AppSSO.

The credentials provided by the service resource claim are:

- `client-id`: the identifier of your `Workload` that AppSSO is registered with. This is a unique identifier.
- `client-secret`: secret string value used by AppSSO to verify your client. Keep this value secret.
- `issuer-uri`: web address of AppSSO `AuthServer` and the primary location that your `Workload` navigates to when interacting with AppSSO.

- `authorization-grant-types`: list of the desired OAuth 2 grant types.
- `client-authentication-method`: method in which the client is authenticated when requesting an identity or access token.
- `scope`: list of the desired scopes that your application's users have access to.

These credentials are mounted onto your `Workload`'s `Pod`(one or more) as individual files at the following locations:

```
/bindings
  /<name-of-service-claim>
    /client-id
    /client-secret
    /issuer-uri
    /authorization-grant-types
    /client-authentication-method
    /scope
```

Following the earlier example, you can find the location of mounted credentials on every `Workload Pod` at:

```
/bindings/my-workload-client
```

Given these auto-generated values, your `Workload` can now load them at runtime and bind to an AppSSO `AuthServer` at start-up time. Reading the values from the file system is left to the implementor as to the approach taken.

Secure a Spring Boot workload

This topic tells you how to secure a sample Spring Boot `Workload` with Application Single Sign-On (commonly called AppSSO), which runs on Tanzu Application Platform (commonly called TAP).

Follow these steps to deploy a sample Spring Boot `Workload`:

1. Get the sample application.
2. Create a namespace for workloads.
3. Apply a client registration.
4. Create a resource claim for the workload.
5. (Optional) Ensure `Workload` trusts `AuthServer`.
6. Deploy the workload.

Get the sample application

Follow these steps to fetch the AppSSO Spring Boot application source code:

1. Download the AppSSO Starter Java accelerator from the TAP GUI accelerators located on your TAP cluster:
    - Option 1: Use the TAP GUI dashboard through browser.

      Navigate to Application Accelerators and download the "AppSSO Starter Java" accelerator.
    - Option 2: Use the Tanzu Accelerator CLI.

      Download the zip file of the accelerator source code by running:

```
tanzu accelerator generate appsso-starter-java --server-url <TAP_GUI_SERV
ER_URL>
```

2. Unzip the resulting `.zip` file into directory `appsso-starter-java` in your workspace.

   ```
   unzip appsso-starter-java
   ```

3. With the resulting project, create an accessible remote Git repository and push your accelerator to the Git remote repository.

Create a namespace for workloads

You must create a namespace for your workloads for the `Workload` resources to function properly. If you have a workloads namespace already, you can skip this step.

```
kubectl create namespace my-apps
kubectl label namespaces my-apps apps.tanzu.vmware.com/tap-ns=""
```

For more information about provisioning namespaces for workloads, see Set up developer namespaces.

Create a `ClientRegistration`

Example: A `ClientRegistration` named `appsso-starter-java` in the `my-apps` namespace. `appsso-starter-java` is attached to an existing `AuthServer` with labels `my-sso=true,env=dev` and an allowance of client registrations from the `my-apps` namespace.

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: appsso-starter-java
  namespace: my-apps
spec:
  authServerSelector:
    matchLabels:
      my-sso: "true"
      env: dev
  clientAuthenticationMethod: client_secret_basic
  authorizationGrantTypes:
  - authorization_code
  redirectURIs:
  - https://appsso-starter-java.my-apps.<TAP_CLUSTER_DOMAIN_NAME>/login/oauth2/code/ap
psso-starter-java
  - http://appsso-starter-java.my-apps.<TAP_CLUSTER_DOMAIN_NAME>/login/oauth2/code/app
sso-starter-java
  scopes:
  - name: openid
```

Note

You can choose `redirectURIs` that use http or https based on your need. The prefix of the `redirectURIs` denotes the name and namespace of the `Workload`. In this case, it is `appsso-starter-java` and `my-apps`. Keep the suffix formatted as `/login/oauth2/code/{ClientRegistration.metadata.name}`. For more information about the redirect URI format of Spring Security OAuth 2 Client library, see Configure AppSSO for workloads.

The accelerator is pre-packaged with a ytt-templated `ClientRegistration` that is located in `client.yaml`. You can generate the same `ClientRegistration` as earlier with your specific values by running:

```
ytt \
  --file client.yaml \
  --data-value namespace=my-apps \
  --data-value workload_name=appsso-starter-java \
  --data-value domain=<TAP_CLUSTER_DOMAIN_NAME> \
  --data-value authserver_label.my-sso=true \
  --data-value authserver_label.env=dev
```

Where:

- `namespace` is the namespace where the workload runs.
- `workload_name` is the distinct instance name of the deployed accelerator.
- `domain` is the domain name where the workload is deployed. The workload instance uses a subdomain to distinguish itself from other workloads. When working within a local cluster, you can use `127.0.0.1.nip.io` to establish a functional DNS route.
- `authserver_label.{matching-label}` is the uniquely identifying label (one or more) for your `AuthServer`. In this example, the `AuthServer` resource is assumed to have labels `my-sso: "true"` and `env: dev`. You can add as many identifying labels as needed.

You can apply the `ClientRegistration` and verify its status by running:

```
kubectl get clientregistration appsso-starter-java --namespace my-apps
```

Claim the `ClientRegistration`

Follow these steps to claim the `ClientRegistration`:

1. Create a service resource claim for the `ClientRegistration` created earlier by using the Tanzu Services plugin CLI:

   ```
   tanzu service resource-claim create appsso-starter-java \
       --namespace my-apps \
       --resource-namespace my-apps \
       --resource-name appsso-starter-java \
       --resource-kind ClientRegistration \
       --resource-api-version "sso.apps.tanzu.vmware.com/v1alpha1"
   ```

2. Check the status of the claim by running:

   ```
   tanzu service claim list --namespace my-apps
   ```

   Expect to see the `appsso-starter-java` claim with the `Ready` status as `True`.

Ensure `Workload` trusts `AuthServer`

For TAP cluster with a custom or self-signed CA certificate, see Configure workloads to trust a custom Certificate Authority (CA).

Deploy the `Workload`

Follow these steps to deploy the `Workload`:

1. Create the Spring Boot accelerator `Workload` by running:

```
tanzu apps workload create appsso-starter-java \
    --namespace my-apps \
    --type web \
    --label app.kubernetes.io/part-of=appsso-starter-java \
    --build-env "BP_JVM_VERSION=17" \
    --service-ref "appsso-starter-java=services.apps.tanzu.vmware.com/v1alpha1:
ResourceClaim:appsso-starter-java" \
    --service-ref "ca-cert=v1:Secret:tap-ca-cert" \
    --git-repo "<GIT_LOCATION_OF_YOUR_ACCELERATOR>" \
    --git-branch main \
    --live-update
```

Important

Although you can assign any name to the `ResourceClaim`, the `Workload`'s service reference name must match the `ClientRegistration` name.

```
--service-ref "**appsso-starter-java**=services.apps.tanzu.vmware.
com/v1alpha1:ResourceClaim:appsso-starter-java"
```

If the service reference name does not match the `ClientRegistration` name, the `Workload` generates a redirect URI that the `AuthServer` will not accept.

It might take a few minutes for the workload to become available through a browser-accessible URL.

2. Query the latest status of the workload by running:

```
tanzu apps workload get appsso-starter-java --namespace my-apps
```

3. Monitor the `Workload` logs:

```
tanzu apps workload tail appsso-starter-java --namespace my-apps
```

After the status of the workload reaches the `Ready` state, you can navigate to the provided URL, which looks similar to:

```
https://appsso-starter-java.my-apps.<TAP_CLUSTER_DOMAIN_NAME>
```

4. Open your preferred web browser and navigate to the URL.

   Expect to see a large log-in button tailored for authenticating with AppSSO.

Cleaning up

Delete the running application by running the following commands:

1. Delete the sample application `Workload`:

```
tanzu apps workload delete appsso-starter-java --namespace my-apps
```

2. Delete the service resource claim for the `ClientRegistration`:

```
tanzu service resource-claim delete appsso-starter-java --namespace my-apps
```

3. Disconnect the client from AppSSO:

```
kubectl delete clientregistration appsso-starter-java --namespace my-apps
```

Secure a single-page app workload

This topic tells you how to to secure a sample single-page Angular app `Workload` with Application Single Sign-On (commonly called AppSSO), which runs on Tanzu Application Platform (commonly known as TAP).

Follow these steps to deploy a sample single-page app `Workload`:

1. Get the sample application.
2. Create a namespace for workloads.
3. Apply a client registration.
4. Verify application authentication settings.
5. Start a sample back end.
6. Deploy the Workload.

Get the sample application

Follow these steps to fetch the single-page Angular app source code:

1. Download the Angular Frontend accelerator from the Tanzu Application Platform GUI accelerators located on your Tanzu Application Platform cluster:
    - Option 1: Use the Tanzu Application Platform GUI dashboard by using a browser.

      Navigate to Application Accelerators and choose the **Angular Frontend** accelerator and then select the **Single Sign-on** option.
    - Option 2: Use the Tanzu Accelerator CLI.

      Download the zip file of the accelerator source code and identify your `AuthServer` Issuer URI by running:

      ```
      kubectl get authserver -A
      ```

      Generate the accelerator by using the `tanzu accelerator` CLI:

      ```
      tanzu accelerator generate angular-frontend \
        --server-url TAP-GUI-SERVER-URL \
        --options '{
          "useSingleSignOn": true,
          "authority": "AUTHSERVER-ISSUER-URI",
          "namespace": "my-apps",
          "authorityLabelKey": "my-sso",
          "authorityLabelValue": "true"
        }'
      ```
2. Unzip the resulting `.zip` file into directory `angular-frontend` in your workspace:

    ```
    unzip angular-frontend
    cd angular-frontend
    git init
    git branch -M main
    git remote add origin YOUR-ACCELERATOR-GIT-REPOSITORY
    git push origin main -u
    ```

For public clients, the `AuthServer` only supports the Authorization Code Flow: `response_type=code`, because public clients such as single page apps cannot safely store sensitive information such as client secrets.

3. Push the resulting directory to the remote Git repository.

Create a namespace for workloads

You must create a namespace for your workloads for the `Workload` resources to function properly. If you have a workloads namespace already, you can skip this step.

```
kubectl create namespace my-apps
kubectl label namespaces my-apps apps.tanzu.vmware.com/tap-ns=""
```

For more information about provisioning namespaces for running `Workloads`, see Set up developer namespaces.

Create a `ClientRegistration`

You must create a `ClientRegistration` to register the frontend application with the `AuthServer`.

Example: A `ClientRegistration` named `angular-frontend` in the `my-apps` namespace. `angular-frontend` is attached to an existing `AuthServer` with labels `my-sso=true` and an allowance of client registrations from the `my-apps` namespace.

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: angular-frontend
  namespace: my-apps
spec:
  authServerSelector:
    matchLabels:
      my-sso: "true"
  clientAuthenticationMethod: none
  authorizationGrantTypes:
  - authorization_code
  redirectURIs:
  - https://angular-frontend.my-apps.TAP-CLUSTER-DOMAIN-NAME/user-profile
  - https://angular-frontend.my-apps.TAP-CLUSTER-DOMAIN-NAME/customer-profiles/list
  - https://angular-frontend.my-apps.TAP-CLUSTER-DOMAIN-NAME/
  - http://angular-frontend.my-apps.TAP-CLUSTER-DOMAIN-NAME/user-profile
  - http://angular-frontend.my-apps.TAP-CLUSTER-DOMAIN-NAME/customer-profiles/list
  - http://angular-frontend.my-apps.TAP-CLUSTER-DOMAIN-NAME/
  scopes:
  - name: openid
  - name: email
  - name: profile
  - name: message.read
  - name: message.write
```

Note

You can choose `redirectURIs` that use HTTP or HTTPS based on your need. The prefix of the `redirectURIs` denotes the name and namespace of the `Workload`. In this scenario, it is `angular-frontend` and `my-apps`.

You can apply the `ClientRegistration` and verify the status is `Ready` by running:

```
kubectl get clientregistration angular-frontend --namespace my-apps
```

Verify application authentication settings

Within the single-page Angular app accelerator, authentication configuration settings are located in `src/assets/auth.conf.json`. After generating the accelerator, expect to observe the populated settings.

Open the file and verify that it adheres to the following structure:

```
{
  "authority": "AUTHSERVER-ISSUER-URI",
  "clientId": "my-apps_angular-frontend",
  "scope": [ "openid", "profile", "email", "message.read", "message.write" ]
}
```

You can retrieve the `clientId` field by running:

```
kubectl get secret angular-frontend -n my-apps -o jsonpath="{.data.client-id}" | base6
4 -d
```

Start a sample back end

Important

You can skip this step if you have a `java-rest-service` back end running already.

The `angular-frontend` sample application requires a back end application to start properly:

1. Start a sample simulated back end by running:

   ```
   kubectl run sample-backend --image nginx:NGINX-VERSION -n my-apps
   kubectl expose pod sample-backend --port 80 -n my-apps
   ```

2. In the `angular-frontend` source code, edit the `.server.location[/api/].proxy_pass` field in the `nginx.conf` file at the root of the source directory.

3. After updating the value, commit the changes to the Git remote repository:

   ```
   server {
    # ...
    location /api/ {
      proxy_pass http://sample-backend.my-apps/api/;
    }
    # ...
   }
   ```

Deploy the `Workload`

Follow these steps to deploy the `Workload`:

1. Create the `angular-frontend` accelerator `Workload` by running:

   ```
   tanzu apps workload create angular-frontend \
       --namespace my-apps \
       --type web \
       --param "clusterBuilder=base" \
   ```

```
    --param "annotations=autoscaling.knative.dev/minScale: \"1\"" \
    --label app.kubernetes.io/part-of=angular-frontend \
    --git-repo "GIT-LOCATION-OF-YOUR-ACCELERATOR" \
    --git-branch main
```

Note

As an alternative approach to creating the workload, you can declaratively define a `Workload` resource by using `config/workload.yaml` within the source repository.

It might take a few minutes for the workload to become available through a browser-accessible URL.

2. Query the latest status of the workload by running:

```
tanzu apps workload get angular-frontend --namespace my-apps
```

3. Monitor the `Workload` logs:

```
tanzu apps workload tail angular-frontend --namespace my-apps
```

After the status of the workload reaches the `Ready` state, you can navigate to the provided URL, which looks similar to:

```
https://angular-frontend.my-apps.TAP-CLUSTER-DOMAIN-NAME/user-profile
```

4. Open your preferred web browser and navigate to the URL.

Expect to be prompted to sign in by using AppSSO. After successfully signing in, the profile page displays your identifying information.

Clean up

Delete the running application by running these commands:

1. Delete the sample application `Workload`:

```
tanzu apps workload delete angular-frontend --namespace my-apps
```

2. Delete the service resource claim for the `ClientRegistration`:

```
tanzu service resource-claim delete angular-frontend --namespace my-apps
```

3. Disconnect the client from AppSSO:

```
kubectl delete clientregistration angular-frontend --namespace my-apps
```

4. Delete the sample back end if was previously applied:

```
kubectl delete svc sample-backend --namespace my-apps
kubectl delete pod sample-backend --namespace my-apps
```

Custom resource definitions (CRDs)

- AuthServer

- ClientRegistration

AuthServer API for AppSSO

In Application Single Sign-On (commonly called AppSSO), `AuthServer` represents the request for an OIDC authorization server. It causes the deployment of an authorization server backed by Redis over mutual TLS if no storage is defined.

An `AuthServer` should have labels which allow to uniquely match it amongst others. `ClientRegistration` selects an `AuthServer` by label selector and needs a unique match to be successful.

To allow `ClientRegistrations` from all or a restricted set of Namespaces, the annotation `sso.apps.tanzu.vmware.com/allow-client-namespaces` must be set. Its value is a comma-separated list of allowed Namespaces, e.g. `"app-team-red,app-team-green"`, or `"*"` if it should allow clients from all namespaces. If the annotation is missing, no clients are allowed.

The issuer URI, which is the point of entry for clients and end-users, is constructed through the package's `domain_template`. You can view the issuer URI by running `kubectl get authserver -n authservers`.

See Issuer URI & TLS for more information.

Token signature keys are configured by using `spec.tokenSignature`. This is a required field. See Token signatures for more context.

You can configure identity providers under `spec.identityProviders`. If there is none, end-users can not log in. For more information about configuring identity providers, see Identity providers.

The deployment can be further customized by configuring replicas, resources, http server and logging properties.

An `AuthServer` reconciles into the following resources in its namespace:

```
AuthServer/my-authserver
├─Certificate/my-authserver-redis-client                    # if no storage is defined
├─Certificate/my-authserver-redis-server                    # if no storage is defined
├─Certificate/my-authserver-root
├─ConfigMap/my-authserver-ca-cert
├─Deployment/my-authserver-auth-server
├─Deployment/my-authserver-redis                            # if no storage is defined
├─Issuer/my-authserver-bootstrap
├─Issuer/my-authserver-root
├─Role/my-authserver-auth-server
├─RoleBinding/my-authserver-auth-server
├─Secret/my-authserver-auth-server-clients
├─Secret/my-authserver-auth-server-keys
├─Secret/my-authserver-auth-server-properties
├─Secret/my-authserver-redis-service-binding                # if no storage is defined
├─Secret/my-authserver-redis-client-cert-keystore-password # if no storage is defined
├─Secret/my-authserver-registry-credentials
├─Service/my-authserver-redis                               # if no storage is defined
└─ServiceAccount/my-authserver-auth-server
```

Spec

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: ""
  namespace: ""
  labels: { } # required, must uniquely identify this AuthServer
```

```
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "" # required, must be "*" or a
comma-separated list of allowed client namespaces
    sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: "" # optional
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: "" # optional
    sso.apps.tanzu.vmware.com/allow-unsafe-cors: "" # optional
spec:
  # .tls is optional if a default issuer is set
  tls:
    # must be one and only one of issuerRef, certificateRef or secretRef, unless deact
ivated
    issuerRef:
      name: ""
      kind: ""
      group: cert-manager.io
    certificateRef:
      name: ""
    secretRef:
      name: ""
    deactivated: false # If true, requires annotation `sso.apps.tanzu.vmware.com/allow
-unsafe-issuer-uri: ""`.
    disabled: false # deprecated, use 'deactivated' instead. If true, requires annotat
ion `sso.apps.tanzu.vmware.com/allow-unsafe-issuer-uri: ""`.
  cors:
    allowOrigins: # optional, cannot be combined with 'allowAllOrigins'.
      - ""
    allowAllOrigins: false # optional
                           # If true, requires annotation `sso.apps.tanzu.vmware.com/a
llow-unsafe-cors: ""`.
                           # Cannot be combined with 'allowOrigins'.
  tokenSignature: # required
    signAndVerifyKeyRef:
      name: "" # Must be a secret that contains an RSA private key with a minimum leng
th of 2048 bits.
    extraVerifyKeyRefs:
      - name: "" # Must be a secret that contains an RSA private key with a minimum le
ngth of 2048 bits.
  storage: # optional
    redis: # required if 'storage' is defined
      serviceRef: # Reference to a provisioned service within the same namespace as th
is AuthServer. Only supports Secret reference.
        apiVersion: "v1"
        kind: "Secret"
        name: ""
  caCerts: # optional
    - secretRef: # Reference to Secret resource within the same namespace as this Auth
Server.
        name: ""
  identityProviders: # optional
    # each must be one and only one of internalUnsafe, ldap, openID or saml
    - name: "" # must be unique
      internalUnsafe: # requires annotation `sso.apps.tanzu.vmware.com/allow-unsafe-id
entity-provider: ""`
        users:
          - username: ""
            password: ""
            givenName: ""
            familyName: ""
            email: ""
            emailVerified: false
            roles:
              - ""
        accessToken: # optional
          scope:
            defaults: # optional
            - ""
```

```
            rolesToScopes: # optional
              - fromRole: ""
                toScopes:
                  - ""
    - name: "" # must be unique
      ldap:
        server:
          scheme: ""
          host: ""
          port: 0
          base: ""
        bind:
          dn: ""
          passwordRef:
            name: ldap-password
        user:
          searchFilter: ""
          searchBase: ""
        roles: # optional
          fromUpstream:
            attribute: "" # required
            search:
              filter: ""
              base: ""
              subTree: false
              depth: 0
          filterBy: # optional
            - exactMatch: ""
            - regex: "" # must be valid regular expression
        accessToken: # optional
          scope:
            defaults: # optional
            - ""
            rolesToScopes: # optional
              - fromRole: ""
                toScopes:
                - ""
        group:    # deprecated, use 'ldap.roles.fromUpstream' instead.
          search: # deprecated, use 'ldap.roles.fromUpstream.search' instead.
            filter: ""
            base: ""
            subTree: false
            depth: 0
          roleAttribute: "" # deprecated, use 'ldap.roles.fromUpstream.attribute' inst
ead.
    - name: "" # must be unique
      openID:
        issuerURI: ""
        clientID: ""
        clientSecretRef:
          name: ""
        scopes:
          - ""
        claimMappings: # deprecated, use 'openID.roles.fromUpstream.claim' instead.
          roles: ""
        roles: # optional
          fromUpstream:
            claim: "" # required
          filterBy: # optional
            - exactMatch: ""
            - regex: "" # must be valid regular expression
        accessToken: # optional
          scope:
            defaults: # optional
            - ""
            rolesToScopes: # optional
```

```
              - fromRole: ""
                toScopes:
                - ""
    - name: "" # must be unique
      saml:
        metadataURI: ""
        claimMappings: { }
        roles: # optional
          fromUpstream:
            attribute: "" # required
          filterBy: # optional
            - exactMatch: ""
            - regex: "" # must be valid regular expressions
        accessToken: # optional
          scope:
            defaults: # optional
              - ""
            rolesToScopes: # optional
              - fromRole: ""
                toScopes:
                  - ""
  replicas: 1 # optional, default 2
  logging: "" # optional, must be valid YAML
  server: "" # optional, must be valid YAML
  resources: # optional, default {requests: {cpu: "256m", memory: "300Mi"}, limits: {c
pu: "2", memory: "768Mi"}}
    requests:
      cpu: ""
      mem: ""
    limits:
      cpu: ""
      mem: ""
  redisResources: # optional, default {requests: {cpu: "50m", memory: "100Mi"}, limit
s: {cpu: "100m", memory: "256Mi"}}
    requests:
      cpu: ""
      mem: ""
    limits:
      cpu: ""
      mem: ""
status:
  observedGeneration: 0
  issuerURI: ""
  clientRegistrationCount: 1
  tokenSignatureKeyCount: 0
  deployments:
    authServer:
      configHash: ""
      image: ""
      replicas: 0
    redis: # leave empty if storage is configured by the service operator
      image: ""
  storage:
    redis:
      host: "" # the hostname of the configured Redis
      port: "" # the port of the configured Redis
  tls:
    deactivated: false
    issuerRef:
      name: ""
      kind: ""
      group: cert-manager.io
  conditions:
    - lastTransitionTime:
      message: ""
      reason: ""
```

```
      status: "True" # or "False"
      type: ""
```

Alternatively, you can interactively discover the spec with:

```
kubectl explain authservers.sso.apps.tanzu.vmware.com
```

Status & conditions

The `.status` subresource helps you to learn the `AuthServer`'s readiness, resulting deployments, attached clients and to troubleshoot issues.

`.status.issuerURI` is the templated issuer URI. This is the entry point for any traffic.

`.status.tls` is the actual TLS configuration.

`.status.tokenSignatureKeyCount` is the number of currently configured token signature keys.

`.status.clientRegistrationCount` is the number of currently registered clients.

`.status.deployments.authServer` describes the current authorization server deployment by listing the running image, its replicas, the hash of the current configuration and the generation which has last resulted in a restart.

`.status.deployments.redis` describes the current provided Redis deployment by listing its running image. This field is nil if storage is defined explicitly by using `.spec.storage`.

`.status.storage.redis` describes the configured Redis storage such as host name and port number.

`.status.conditions` documents each step in the reconciliation:

- `Valid`: Is the spec valid?
- `ImagePullSecretApplied`: Has the image pull secret been applied?
- `SignAndVerifyKeyResolved`: Has the single sign-and-verify key been resolved?
- `ExtraVerifyKeysResolved`: Have the single extra verify keys been resolved?
- `IdentityProvidersResolved`: Has all identity provider configuration been resolved?
- `ConfigResolved`: Has the complete configuration for the authorization server been resolved?
- `AuthServerConfigured`: Has the complete configuration for the authorization server been applied?
- `IssuerURIReady`: Is the authorization server yet responding to `{.status.issuerURI}/.well-known/openid-configuration`?
- `Ready`: whether all the previous conditions are “True”

The super condition `Ready` denotes a fully successful reconciliation of a given `ClientRegistration`.

If everything goes well you will see something like this:

```
issuerURI: "https://..."
observedGeneration: 1
tokenSignatureKeyCount: 0
clientRegistrationCount: 0
caCerts:
  - cert:
      subject: ""
    source:
      secretEntry: ""
deployments:
  authServer:
```

```
    LastParentGenerationWithRestart: 1
    configHash: "11216479096262796218"
    image: "..."
    replicas: 1
  redis: # leave empty if external storage is defined
    image: "..."
storage:
  redis:
   host: "" # the host name of the configured Redis
   port: "" # the port of the configured Redis
tls:
  deactivated: false
  # One of issuerRef, certificateRef or secretRef is set if TLS is enabled
  issuerRef:
    name: ""
    kind: ""
    group: ""
  certificateRef:
    name: ""
  secretRef:
    name: ""
conditions:
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: KeysConfigSecretUpdated
    status: "True"
    type: AuthServerConfigured
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: Resolved
    status: "True"
    type: ConfigResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: ExtraVerifyKeysResolved
    status: "True"
    type: ExtraVerifyKeysResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: Resolved
    status: "True"
    type: IdentityProvidersResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: ImagePullSecretApplied
    status: "True"
    type: ImagePullSecretApplied
  - lastTransitionTime: "2022-08-24T09:58:28Z"
    message: ""
    reason: Ready
    status: "True"
    type: IssuerURIReady
  - lastTransitionTime: "2022-08-24T09:58:28Z"
    message: ""
    reason: Ready
    status: "True"
    type: Ready
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: SignAndVerifyKeyResolved
    status: "True"
    type: SignAndVerifyKeyResolved
  - lastTransitionTime: "2022-08-24T09:58:10Z"
    message: ""
    reason: Valid
```

```
    status: "True"
    type: Valid
```

RBAC

The `ServiceAccount` of the authorization server has a `Role` with the following permissions:

```
- apiGroups:
    - ""
  resources:
    - secrets
  verbs:
    - get
    - list
    - watch
  resourceNames:
    - { name }-auth-server-keys
    - { name }-auth-server-clients
```

Example

This example requests an authorization server with two token signature keys and two identity providers.

Note

The label used for matching to ClientRegistrations must be unique across namespaces.

```
---
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: AuthServer
metadata:
  name: authserver-sample
  namespace: default
  labels:
    identifier: authserver-identifier
    sample: "true"
  annotations:
    sso.apps.tanzu.vmware.com/allow-client-namespaces: "*"
    sso.apps.tanzu.vmware.com/allow-unsafe-identity-provider: ""
    sso.apps.tanzu.vmware.com/allow-unsafe-cors: ""
spec:
  replicas: 1
  tls:
    issuerRef:
      name: my-cluster-issuer
      kind: ClusterIssuer
  tokenSignature:
    signAndVerifyKeyRef:
      name: sample-token-signing-key
    extraVerifyKeyRefs:
      - name: sample-token-verification-key
  cors:
    allowAllOrigins: true
  identityProviders:
    - name: internal
      internalUnsafe:
        users:
          - username: user
```

```
            password: password
            roles:
              - message.write
    - name: okta
      openID:
        issuerURI: https://dev-xxxxxx.okta.com
        clientID: xxxxxxxxxxxx
        clientSecretRef:
          name: okta-client-secret
        authorizationUri: https://dev-xxxxxx.okta.com/oauth2/v1/authorize
        tokenUri: https://dev-xxxxxx.okta.com/oauth2/v1/token
        jwksUri: https://dev-xxxxxx.okta.com/oauth2/v1/keys
        scopes:
          - openid
        roles:
          fromUpstream:
            claim: my_custom_okta_roles_claim
        accessToken:
          scope:
            defaults:
              - "developer.read"
              - "developer.write"
            rolesToScopes:
              - fromRole: "finance"
                toScopes:
                  - "finance.read"
                  - "finance.write"

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: sample-token-signing-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)

---
apiVersion: secretgen.k14s.io/v1alpha1
kind: RSAKey
metadata:
  name: sample-token-verification-key
  namespace: default
spec:
  secretTemplate:
    type: Opaque
    stringData:
      key.pem: $(privateKey)
      pub.pem: $(publicKey)

---
apiVersion: v1
kind: Secret
metadata:
  name: okta-client-secret
  namespace: default
stringData:
  clientSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

ClientRegistration API for AppSSO

In Application Single Sign-On (commonly called AppSSO), `ClientRegistration` is the request for client credentials for an AuthServer.

It implements the Service Bindings `ProvisionedService`. The credentials are returned as a Service Bindings `Secret`.

A `ClientRegistration` must uniquely identify an `AuthServer` by using `spec.authServerSelector`. If it matches none, too many or a disallowed `AuthServer`, it does not get credentials. The other fields are for the configuration of the client on the `AuthServer`.

Spec

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: ""
  namespace: ""
spec:
  authServerSelector: # required
    matchLabels: { }
  redirectURIs: # required
    - ""
  scopes: # optional
    - name: ""
      description: ""
  authorizationGrantTypes: # optional
    - client_credentials
    - authorization_code
    - refresh_token
  clientAuthenticationMethod: "" # optional, values accepted are described in Client a
uthentication methods section
  requireUserConsent: false # optional
status:
  authServerRef:
    apiVersion: ""
    issuerURI: ""
    kind: ""
    name: ""
    namespace: ""
  binding:
    name: ""
  clientID: ""
  clientSecretHelp: ""
  conditions:
    - lastTransitionTime: ""
      message: ""
      reason: ""
      status: "True" # or "False"
      type: ""
  observedGeneration: 0
```

Alternatively, you can interactively discover the spec with:

```
kubectl explain clientregistrations.sso.apps.tanzu.vmware.com
```

Client authentication methods

Client authentication methods supported by `ClientRegistration` resource are:

- `client_secret_basic`: HTTP header based client authentication (default).
- `client_secret_post`: HTTP POST body based client authentication.

- `basic` (deprecated): HTTP header based client authentication. Use `client_secret_basic` instead.
- `post` (deprecated): HTTP POST body based client authentication. Use `client_secret_post` instead.
- `none`: No client authentication. Required for public clients. For more information, see Public clients and CORS.

Caution

When running Workloads using Spring Boot 3, you must use `client_secret_basic` or `client_secret_post`. For more information, see Spring Boot 3 based Workloads and ClientRegistration resources.

Status & conditions

The `.status` subresource helps you to learn about your client credentials, the matched `AuthServer` and to troubleshoot issues.

`.status.authServerRef` identifies the successfully matched `AuthServer` and its issuer URI.

`.status.binding.name` is the name of the Service Bindings `Secret` which contains the client credentials.

`.status.conditions` documents each step in the reconciliation:

- `Valid`: Is the spec valid?
- `AuthServerResolved`: Has the targeted `AuthServer` been resolved?
- `ClientSecretResolved`: Has the client secret been resolved?
- `ServiceBindingSecretApplied`: Has the Service Bindings Secret with the client credentials been applied?
- `AuthServerConfigured`: Has the resolved `AuthServer` been configured with the client?
- `Ready`: whether all the previous conditions are “True”

The super condition `Ready` denotes a fully successful reconciliation of a given `ClientRegistration`.

If everything goes well you will see something like this:

```
status:
  authServerRef:
    apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
    issuerURI: http://authserver-sample.default
    kind: AuthServer
    name: authserver-sample
    namespace: default
  binding:
    name: clientregistration-sample
  clientID: default_clientregistration-sample
  clientSecretHelp: 'Find your clientSecret: ''kubectl get secret clientregistration-s
ample --namespace default'''
  conditions:
    - lastTransitionTime: "2022-05-13T07:56:41Z"
      message: ""
      reason: Updated
      status: "True"
      type: AuthServerConfigured
    - lastTransitionTime: "2022-05-13T07:56:40Z"
      message: ""
```

```
      reason: Resolved
      status: "True"
      type: AuthServerResolved
    - lastTransitionTime: "2022-05-13T07:56:40Z"
      message: ""
      reason: ResolvedFromBindingSecret
      status: "True"
      type: ClientSecretResolved
    - lastTransitionTime: "2022-05-13T07:56:41Z"
      message: ""
      reason: Ready
      status: "True"
      type: Ready
    - lastTransitionTime: "2022-05-13T07:56:40Z"
      message: ""
      reason: Applied
      status: "True"
      type: ServiceBindingSecretApplied
    - lastTransitionTime: "2022-05-13T07:56:40Z"
      message: ""
      reason: Valid
      status: "True"
      type: Valid
  observedGeneration: 1
```

Example

```
apiVersion: sso.apps.tanzu.vmware.com/v1alpha1
kind: ClientRegistration
metadata:
  name: my-client-registration
  namespace: app-team
spec:
  authServerSelector:
    matchLabels:
      for: app-team
      ldap: "true"
  redirectURIs:
    - "https://127.0.0.1:8080/authorized"
    - "https://my-application.com/authorized"
  requireUserConsent: false
  clientAuthenticationMethod: client_secret_basic
  authorizationGrantTypes:
    - "client_credentials"
    - "refresh_token"
  scopes:
    - name: "openid"
      description: "To indicate that the application intends to use OIDC to verify the
user's identity"
    - name: "email"
      description: "The user's email"
    - name: "profile"
      description: "The user's profile information"
```

The client is registered with the authorization server with the given `spec`. The resulting client credentials are available in a `Secret` that the `ClientRegistration` owns.

```
apiVersion: v1
kind: Secret
type: servicebinding.io/oauth2
metadata:
  name: my-client-registration
  namespace: app-team
```

```
data: # fields below are base64-decoded for display purposes only
  type: oauth2
  provider: appsso
  client-id: default_my-client-registration
  client-secret: c2VjcmV0 # auto-generated
  issuer-uri: https://appsso.example.com
  client-authentication-method: basic
  scope: openid,email,profile
  authorization-grant-types: client_credentials,refresh_token
```

Troubleshoot Application Single Sign-on

This topic tells you how to troubleshoot Application Single Sign-On (commonly called AppSSO).

Why is my AuthServer not working?

Generally, `AuthServer.status` is designed to provide you with helpful feedback to debug a faulty `AuthServer`.

Find all `AuthServer` related Kubernetes resources

Identify all `AuthServer` components with Kubernetes common labels. For more information, see Kubernetes documentation.

Query all related `AuthServer` sub-resources by using `app.kubernetes.io/part-of` label. For example:

```
kubectl get all,ingress,service -A -l app.kubernetes.io/part-of=<authserver-name>
```

Logs of all AuthServers

With stern you can tail the logs of all AppSSO managed `Pods` inside your cluster with:

```
stern --all-namespaces --selector=app.kubernetes.io/managed-by=sso.apps.tanzu.vmware.com
```

Change propagation

When applying changes to an `AuthServer`, keep in mind that changes to issuer URI, IDP, server and logging configuration take a moment to be effective as the operator will roll out the authorization server `Deployment`.

Misconfigured `clientSecret`

Problem:

When attempting to sign in, you see `This commonly happens due to an incorrect [client_secret].` It might be because the client secret of an identity provider is misconfigured.

Solution:

Validate the `AuthServer.spec.openid.clientSecretRef`.

Misconfigured redirect URI

Problem:

You see `Error: [invalid_request] OAuth 2.0 Parameter: redirect_uri` when signing in.

Solution:

The `redirectURIs` of a `ClientRegistration` must refer to the URI (one or more) of the registered `Workload`. It does not refer to the URI of the AuthServer. For more information, see Redirect URIs.

Unsupported `id_token_signed_response_alg` with openid `identityProviders`

Problem:

When trying to log in with an OpenID Connect `identityProvider`, you are unable to sign in and observe the following error in the logs:

```
[invalid_id_token] An error occurred while attempting to decode the Jwt: Signed JWT re
jected: Another algorithm expected, or no matching key(s) found.
```

Solution:

Verify the `identityProvider`'s discovery endpoint at `ISSUER-URI/.well-known/openid-configuration` where `ISSUER-URI` is the value set at `spec.identityProviders.openid.issuerURI`.

The value of `id_token_signing_alg_values_supported` must include `RS256`. If it is not in the list, your identity configuration might not support AppSSO.

If `RS256` is present, expect to see a `jwks_uri` key in the discovery endpoint. If you visit the URL stored in this key, it must return at least one RSA key. Otherwise, your identity provider might be misconfigured.

Refer to your identity provider's documentation to enable `RS256` token signing.

Misconfigured identity provider clientSecret

Problem:

- When attempting to sign in, you see `<WORKLOAD_URL> redirected you too many times.` It might be because the client secret of an identity provider is misconfigured.
- If you have access to the authserver logs, verify if there is an entry with the text `"error":"[invalid_client] Client authentication failed: client_secret"`.

Solution:

Validate the secret referenced by the `clientSecretRef` for this particular identity provider in your `authserver.spec`.

Missing scopes

Problem:

When attempting to fetch data after signing in to your application by using AppSSO, you see `[invalid_scope] OAuth 2.0 Parameter: scope`.

Solution:

Add the required scopes into your `ClientRegistration` yaml under `spec.scopes`.

Changes to the secret do not propagate to the `ClientRegistration`. If you recreated the `Secret` that contains the `clientSecret`, you must re-deploy the `ClientRegistration`.

Misconfigured `sub` claim

Problem:

The `sub` claims in `id_token`s and `access_token`s follow the `<providerId>_<userId>` pattern. The previous `<providerId>/<userId>` pattern might cause bugs in URLs without proper URL-encoding in client applications.

Solution:

If your client application stores `sub` claims, you must update the `sub` claims to match the new pattern `<providerId>_<userId>`.

Known Issues

Application Single Sign-On (commonly called AppSSO) has the following known issues.

Unregistration by deletion

You can only deregister an existing, ready `ClientRegistration` from its selected `AuthServer` by deleting it. Breaking the match between the two resources by updating either the labels of the `AuthServer` or the label selector on the `ClientRegistration` does not deregister the client from the authorization server.

Limited number of `ClientRegistrations` per `AuthServer`

The number of `ClientRegistration` for an `AuthServer` is limited at **~2,000**. This is a soft limitation, and if you are attempting to apply more `ClientRegistration` resources than the limit, we cannot guarantee those clients applied past the limit to be in working order. This is subject to change in future product versions.

LetsEncrypt: domain name for Issuer URI limited to 64 characters maximum

If using LetsEncrypt to issue TLS certificates for an `AuthServer`, the domain name for the Issuer URI (excluding the `http{s}` prefix) cannot exceed 64 characters in length. If exceeded, you may receive a LetsEncrypt-specific error during Certificate issuance process. This limitation may be observed when your base domain and subdomain joined together exceed the maximum limit.

Workaround - if your default Issuer URI is too long, utilize the `domain_template` field in AppSSO values yaml to potentially shorten the domain.

For example, you may forgo the namespace in the Issuer URI like so:

```
domain_template: "{{.Name}}.{{.Domain}}"
```


Caution

By leaving out the namespace in your domain template, application routes might conflict if there are multiple `AuthServer`s with the same name but in different

namespaces.

Spring Boot 3 based `Workload`s and `ClientRegistration` resources

If you run a `Workload` based on Spring Boot 3 or use Spring Security OAuth2 Client 3 library in conjunction with `ResourceClaim`s, you must configure your `ClientRegistration` resource to use either of the following client authentication methods:

- `client_secret_basic` (default)
- `client_secret_post`

The existing `post` and `basic` values do not work with Spring Boot 3 based `Workloads` with Spring Cloud Bindings and are deprecated.

Overview of Default roles for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) v1.5 includes:

- Six default roles to help you set up permissions for users and service accounts within a namespace on a cluster that runs one of the Tanzu Application Platform profiles.
- A Tanzu CLI RBAC (role-based access control) plug-in for role binding. For more information, see Bind a user or group to a default role.
- Documentation for integrating with your existing identity management solution.

Default roles

Four roles are for users:

- app-editor
- app-viewer
- app-operator
- service-operator

Two roles are for service accounts associated with the Tanzu Supply Chain:

- workload
- deliverable

The default roles provide an opinionated starting point for the most common permissions that users need when using Tanzu Application Platform. However, as described in the Kubernetes documentation about RBAC, you can create customized roles and permissions that better meet your needs. Aggregated cluster roles are used to build VMware Tanzu Application Platform default roles.

Cluster admins must be careful when creating Roles or ClusterRoles. When changing roles or adding new roles that carry one of the labels used by the default roles, the roles are automatically updated to the aggregation state. It can lead to unintentional changes in functions and permissions to all users.

The default roles are installed with every Tanzu Application Platform profile except for `view`. For an overview of the different roles and their permissions, see Role Descriptions.

Working with roles using the RBAC CLI plug-in

For more information about working with roles, see Bind a user or group to a default role.

Disclaimer

Tanzu Application Platform GUI does not make use of the described roles. Instead, it provides the user with view access for each cluster.

Overview of Default roles for Tanzu Application Platform

Tanzu Application Platform (commonly known as TAP) v1.5 includes:

- Six default roles to help you set up permissions for users and service accounts within a namespace on a cluster that runs one of the Tanzu Application Platform profiles.
- A Tanzu CLI RBAC (role-based access control) plug-in for role binding. For more information, see Bind a user or group to a default role.
- Documentation for integrating with your existing identity management solution.

Default roles

Four roles are for users:

- app-editor
- app-viewer
- app-operator
- service-operator

Two roles are for service accounts associated with the Tanzu Supply Chain:

- workload
- deliverable

The default roles provide an opinionated starting point for the most common permissions that users need when using Tanzu Application Platform. However, as described in the Kubernetes documentation about RBAC, you can create customized roles and permissions that better meet your needs. Aggregated cluster roles are used to build VMware Tanzu Application Platform default roles.

Cluster admins must be careful when creating Roles or ClusterRoles. When changing roles or adding new roles that carry one of the labels used by the default roles, the roles are automatically updated to the aggregation state. It can lead to unintentional changes in functions and permissions to all users.

The default roles are installed with every Tanzu Application Platform profile except for `view`. For an overview of the different roles and their permissions, see Role Descriptions.

Working with roles using the RBAC CLI plug-in

For more information about working with roles, see Bind a user or group to a default role.

Disclaimer

Tanzu Application Platform GUI does not make use of the described roles. Instead, it provides the user with view access for each cluster.

Set up authentication for your Tanzu Application Platform deployment

There are multiple ways to set up authentication for your Tanzu Application Platform (commonly known as TAP) deployment. You can manage authentication at the infrastructure level with your Kubernetes provider, such as Tanzu Kubernetes Grid, EKS, AKS, or GKE.

VMware recommends Pinniped for integrating your identity management into Tanzu Application Platform on multicloud. It provides many supported integrations for widely used identity providers. To use Pinniped, see Installing Pinniped on Tanzu Application Platform and Log in by using Pinniped.

See Integrating Azure Active Directory for Azure Active Directory Integration.

Tanzu Kubernetes Grid

For Tanzu Kubernetes Grid clusters, Pinniped is the default identity solution and is installed as a core package. For more information, see Core Packages and Enable Identity Management in an Existing Deployment in the Tanzu Kubernetes Grid documentation.

Set up authentication for your Tanzu Application Platform deployment

There are multiple ways to set up authentication for your Tanzu Application Platform (commonly known as TAP) deployment. You can manage authentication at the infrastructure level with your Kubernetes provider, such as Tanzu Kubernetes Grid, EKS, AKS, or GKE.

VMware recommends Pinniped for integrating your identity management into Tanzu Application Platform on multicloud. It provides many supported integrations for widely used identity providers. To use Pinniped, see Installing Pinniped on Tanzu Application Platform and Log in by using Pinniped.

See Integrating Azure Active Directory for Azure Active Directory Integration.

Tanzu Kubernetes Grid

For Tanzu Kubernetes Grid clusters, Pinniped is the default identity solution and is installed as a core package. For more information, see Core Packages and Enable Identity Management in an Existing Deployment in the Tanzu Kubernetes Grid documentation.

Install Pinniped on Tanzu Application Platform

Pinniped is used to support authentication on Tanzu Application Platform (commonly known as TAP). This topic tells you how to install Pinniped on a single cluster of Tanzu Application Platform.

Note

This topic only provides an example of one possible installation method for Pinniped on Tanzu Application Platform by using the default Contour ingress controler included in the platform. See Pinniped documentation for more information about the specific installation method that suits your environment.

Use this topic to learn how to deploy two Pinniped components into the cluster:

- **Pinniped Supervisor:** An OIDC server which allows users to authenticate with an external identity provider (IDP). It hosts an API for the concierge component to fulfill authentication requests.
- **Pinniped Concierge:** A credential exchange API that takes a credential from an identity source, for example, Pinniped Supervisor, proprietary IDP, as input. The Pinniped Concierge authenticates the user by using the credential, and returns another credential that is parsable by the host Kubernetes cluster or by an impersonation proxy that acts on behalf of the user.

Prerequisites

Meet these prerequisites:

- Install the package `certmanager`. This is included in Tanzu Application Platform.
- Install the package `contour`. This is included in Tanzu Application Platform.
- Create a `workspace` directory to function as your workspace.

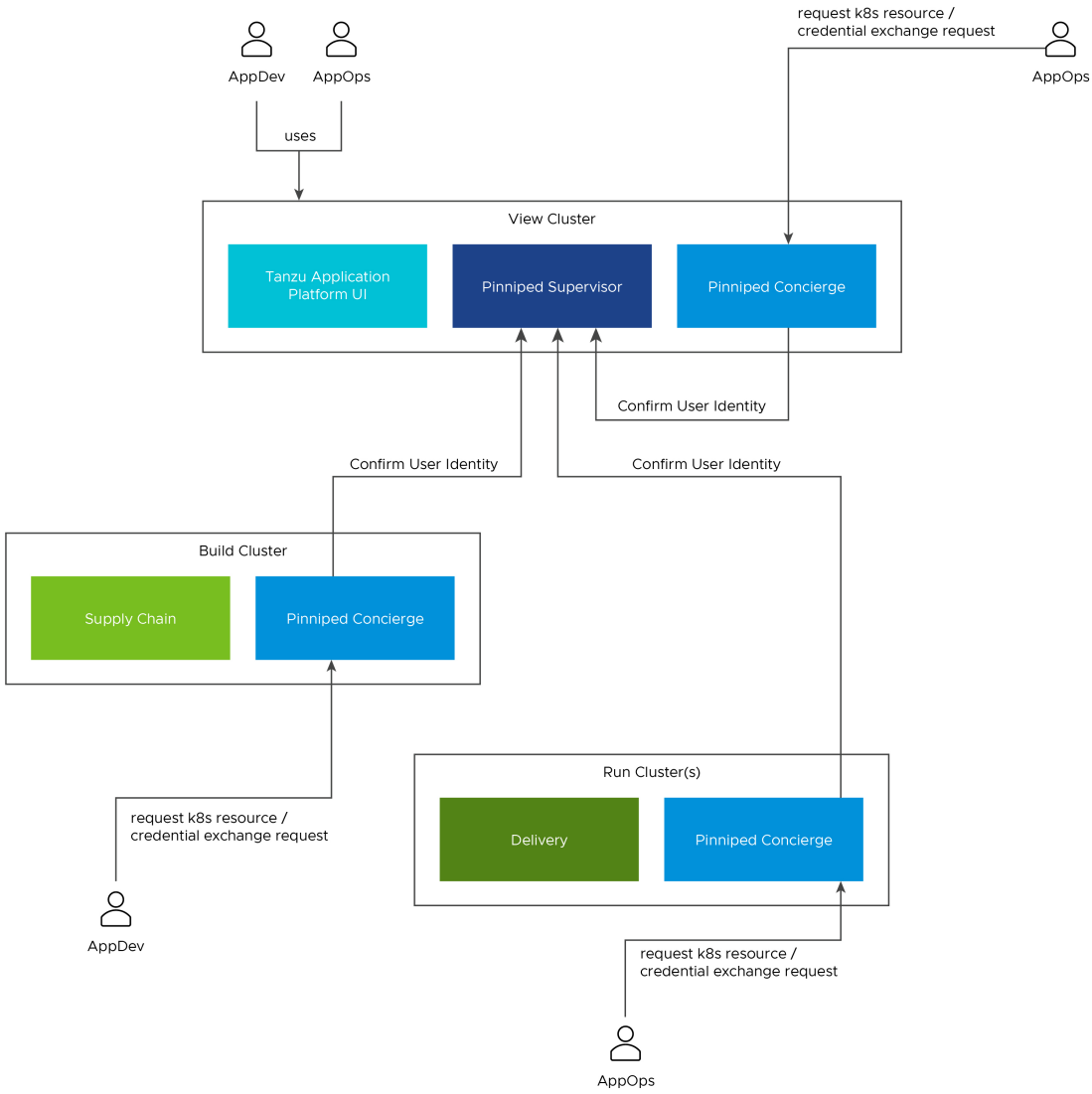
Environment planning

If you run Tanzu Application Platform on a single cluster, both Pinniped Supervisor and Pinniped Concierge are installed to this cluster.

When running a multicluster setup, you must decide which cluster to deploy the Supervisor onto. Furthermore, every cluster must have the Concierge deployed. Pinniped Supervisor runs as a central component that is consumed by multiple Pinniped Concierge instances. As a result, Pinniped Supervisor must be deployed to a single cluster that meets the prerequisites. You can deploy Pinniped Supervisor to the View Cluster of your Tanzu Application Platform, because it is a central single instance cluster. For more information, see Overview of multicluster Tanzu Application Platform.

You must deploy the Pinniped Concierge to every cluster that you want to enable authentication for, including the View Cluster itself.

See the following diagram for a possible deployment model:



For more information about the Pinniped architecture and deployment model, see Pinniped documentation.

Install Pinniped Supervisor by using Let's Encrypt

Follow these steps to install `pinniped-supervisor`:

1. Switch tooling to the desired cluster.
2. Create the necessary certificate files.
3. Create the Ingress resources.
4. Create the `pinniped-supervisor` configuration.
5. Apply these resources to the cluster.

Create Certificates (`letsencrypt` or `cert-manager`)

Choose a fully qualified domain name (FQDN) that can resolve to the Contour instance in the `tanzu-system-ingress` namespace. The FQDN `pinniped-supervisor.example.com` is used in the following sections.

Create a `ClusterIssuer` for `letsencrypt` and a TLS certificate resource for Pinniped Supervisor by creating the following resources and saving them into `workspace/pinniped-`

`supervisor/certificates.yaml`:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
  namespace: cert-manager
spec:
  acme:
    email: "EMAIL"
    privateKeySecretRef:
      name: letsencrypt-staging
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    solvers:
    - http01:
        ingress:
          class: contour

---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: pinniped-supervisor-cert
  namespace: pinniped-supervisor
spec:
  secretName: pinniped-supervisor-tls-cert
  dnsNames:
  - "DNS-NAME"
  issuerRef:
    name: letsencrypt-staging
    kind: ClusterIssuer
```

Where:

- `EMAIL` is the user email address for `letsencrypt`. For example, `your-mail@example.com`
- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`

Create Ingress resources

Create a Service and Ingress resource to make the `pinniped-supervisor` accessible from outside the cluster.

To do so, create the following resources and save them into `workspace/pinniped-supervisor/ingress.yaml`:

```
---
apiVersion: v1
kind: Service
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  ports:
  - name: pinniped-supervisor
    port: 8443
    protocol: TCP
    targetPort: 8443
  selector:
    app: pinniped-supervisor

---
apiVersion: projectcontour.io/v1
```

```
kind: HTTPProxy
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  virtualhost:
    fqdn: "DNS-NAME"
    tls:
      passthrough: true
  routes:
  - services:
    - name: pinniped-supervisor
      port: 8443
```

Where:

- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`
- `tls.passthrough: true` specifies that the TLS connection is forwarded to and terminated in the supervisor pod.

Create the `pinniped-supervisor` configuration

Create a `FederationDomain` to link the concierge to the supervisor instance and configure an `OIDCIdentityProvider` to connect the supervisor to your OIDC Provider. The following example uses `auth0` as the `OIDCIdentityProvider`. For more information about how to configure different identity providers, including OKTA, GitLab, OpenLDAP, Dex, Microsoft AD and more, see Pinniped documentation.

To create the `pinniped-supervisor` configuration, create the following resources and save them into `workspace/pinniped-supervisor/oidc_identity_provider.yaml`:

```
apiVersion: idp.supervisor.pinniped.dev/v1alpha1
kind: OIDCIdentityProvider
metadata:
  namespace: pinniped-supervisor
  name: auth0
spec:
  # Specify the upstream issuer URL associated with your auth0 application.
  issuer: https://"APPLICATION-SUBDOMAIN".auth0.com/

  # Specify how to form authorization requests.
  authorizationConfig:
    additionalScopes: ["openid", "email"]
    allowPasswordGrant: false

  # Specify how claims are mapped to Kubernetes identities. This varies by provider.
  claims:
    username: email
    groups: groups

  # Specify the name of the Kubernetes Secret that contains your
  # application's client credentials (created as follows).
  client:
    secretName: auth0-client-credentials

---
apiVersion: v1
kind: Secret
metadata:
  namespace: pinniped-supervisor
  name: auth0-client-credentials
type: secrets.pinniped.dev/oidc-client
```

```
stringData:
  clientID: "AUTH0-CLIENT-ID"
  clientSecret: "AUTH0-CLIENT-SECRET"

---
apiVersion: config.supervisor.pinniped.dev/v1alpha1
kind: FederationDomain
metadata:
  name: pinniped-supervisor-federation-domain
  namespace: pinniped-supervisor
spec:
  issuer: "DNS-NAME"
  tls:
    secretName: pinniped-supervisor-tls-cert
```

Where:

- `APPLICATION-SUBDOMAIN` is the application specific subdomain that is assigned after the application registration.
- `AUTH0-CLIENT-ID` and `AUTH0-CLIENT-SECRET` are the credentials retrieved from the application registration.
- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`

Apply the resources

After creating the resource files, you can install them into the cluster. Follow these steps to deploy them as a kapp application:

1. Install the `pinniped-supervisor` by running:

   ```
   kapp deploy -y --app pinniped-supervisor -f pinniped-supervisor -f https://get.
   pinniped.dev/v0.22.0/install-pinniped-supervisor.yaml
   ```

   Note

   To keep the security patches up to date, you must install the most recent version of Pinniped. See Vmware Tanzu Pinniped Releases in GitHub for more information.

2. Get the external IP address of Ingress by running:

   ```
   kubectl -n tanzu-system-ingress get svc/envoy -o jsonpath='{.status.loadBalance
   r.ingress[0].ip}'
   ```

3. If not already covered by the Tanzu Application Platform wildcard DNS entry, add an entry to the DNS system to bind the external IP address with.

Switch to production issuer (`letsencrypt` or `cert-manager`)

Follow these steps to switch to a `letsencrypt` production issuer so the generated TLS certificate is recognized as valid by web browsers and clients:

1. Edit the ClusterIssuer for `letsencrypt` and add TLS certificate resource for `pinniped-supervisor` by creating or updating the following resources and saving them into `workspace/pinniped-supervisor/certificates.yaml`:

```
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
  namespace: cert-manager
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: "EMAIL"
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: contour

---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: pinniped-supervisor-cert
  namespace: pinniped-supervisor
spec:
  secretName: pinniped-supervisor-tls-cert
  dnsNames:
  - "DNS-NAME"
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
```

Where:

- `EMAIL` is the user email address for `letsencrypt`. For example, `your-mail@example.com`
- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`

2. Create or update the `pinniped-supervisor` kapp application:

```
kapp deploy -y --app pinniped-supervisor -f pinniped-supervisor -f https://get.
pinniped.dev/v0.22.0/install-pinniped-supervisor.yaml
```

Install Pinniped Supervisor Private CA

Follow these steps to install `pinniped-supervisor`:

1. Switch tooling to the desired cluster.
2. Create the necessary certificate files.
3. Create the Ingress resources.
4. Create the `pinniped-supervisor` configuration.
5. Apply these resources to the cluster.

Create Certificate Secret

Choose a fully qualified domain name (FQDN) that can resolve to the Contour instance in the `tanzu-system-ingress` namespace. Create a certificate by using a CA that the clients trust. This FQDN can be under the `ingress_domain` in the TAP values file, or a dedicated DNS entry. The FQDN `pinniped-supervisor.example.com` is used in the following sections.

After the certificate files are available, they must be encoded to base64 format in a single-line layout. For example, you can encode the certificate file `my.crt` by running:

```
cat my.crt | base64 -w 0
```

Create the following resource and save it into `workspace/pinniped-supervisor/ingress.yaml`:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: pinniped-supervisor-tls-cert
  namespace: pinniped-supervisor
type: kubernetes.io/tls
data:
  tls.crt: PRIVATE-KEY
  tls.key: PUBLIC-KEY
```

Where:

- `PRIVATE-KEY` is the base64 encoded public key.
- `PUBLIC-KEY` is the base64 encoded public key.

Create Ingress resources

Create a Service and Ingress resource to make the `pinniped-supervisor` accessible from outside the cluster.

To do so, create the following resources and save them into `workspace/pinniped-supervisor/ingress.yaml`:

```
---
apiVersion: v1
kind: Service
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  ports:
  - name: pinniped-supervisor
    port: 8443
    protocol: TCP
    targetPort: 8080
  selector:
    app: pinniped-supervisor

---
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: pinniped-supervisor
  namespace: pinniped-supervisor
spec:
  virtualhost:
    fqdn: "DNS-NAME"
    tls:
      passthrough: true
  routes:
  - services:
    - name: pinniped-supervisor
      port: 8443
```

Where:

- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`
- `tls.passthrough: true` specifies that the TLS connection is forwarded to and terminated in the supervisor pod.

Create the `pinniped-supervisor` configuration

Create a `FederationDomain` to link the concierge to the supervisor instance and configure an `OIDCIdentityProvider` to connect the supervisor to your OIDC Provider. The following example uses `auth0` as the `OIDCIdentityProvider`. For more information about how to configure different identity providers, including OKTA, GitLab, OpenLDAP, Dex, Microsoft AD and more, see Pinniped documentation.

To create the `pinniped-supervisor` configuration, create the following resources and save them into `workspace/pinniped-supervisor/oidc_identity_provider.yaml`:

```
apiVersion: idp.supervisor.pinniped.dev/v1alpha1
kind: OIDCIdentityProvider
metadata:
  namespace: pinniped-supervisor
  name: auth0
spec:
  # Specify the upstream issuer URL associated with your auth0 application.
  issuer: https://"APPLICATION-SUBDOMAIN".auth0.com/

  # Specify how to form authorization requests.
  authorizationConfig:
    additionalScopes: ["openid", "email"]
    allowPasswordGrant: false

  # Specify how claims are mapped to Kubernetes identities. This varies by provider.
  claims:
    username: email
    groups: groups

  # Specify the name of the Kubernetes Secret that contains your
  # application's client credentials (created as follows).
  client:
    secretName: auth0-client-credentials

---
apiVersion: v1
kind: Secret
metadata:
  namespace: pinniped-supervisor
  name: auth0-client-credentials
type: secrets.pinniped.dev/oidc-client
stringData:
  clientID: "AUTH0-CLIENT-ID"
  clientSecret: "AUTH0-CLIENT-SECRET"

---
apiVersion: config.supervisor.pinniped.dev/v1alpha1
kind: FederationDomain
metadata:
  name: pinniped-supervisor-federation-domain
  namespace: pinniped-supervisor
spec:
  issuer: "DNS-NAME"
  tls:
    secretName: pinniped-supervisor-tls-cert
```

Where:

- `APPLICATION-SUBDOMAIN` is the application specific subdomain that is assigned after the application registration.
- `AUTH0-CLIENT-ID` and `AUTH0-CLIENT-SECRET` are the credentials retrieved from the application registration.
- `DNS-NAME` is the domain in which the pinniped-supervisor is published. For example, `pinniped-supervisor.example.com`

Apply the resources

After creating the resource files, you can install them into the cluster. Follow these steps to deploy them as a kapp application:

1. Install the supervisor by running:

```
kapp deploy -y --app pinniped-supervisor -f pinniped-supervisor -f https://get.
pinniped.dev/v0.22.0/install-pinniped-supervisor.yaml
```

> **Note**
>
> To keep the security patches up to date, you must install the most recent version of Pinniped. See Vmware Tanzu Pinniped Releases in GitHub for more information.

2. Get the external IP address of Ingress by running:

```
kubectl -n tanzu-system-ingress get svc/envoy -o jsonpath='{.status.loadBalance
r.ingress[0].ip}'
```

3. If not already covered by a Tanzu Application Platform wildcard DNS entry, add an entry to the DNS system to bind the external IP address with.

Install Pinniped Concierge

To install Pinniped Concierge:

1. Switch tooling to the desired cluster.
2. Deploy the Pinniped Concierge by running:

```
kapp deploy -y --app pinniped-concierge \
  -f https://get.pinniped.dev/v0.22.0/install-pinniped-concierge.yaml
```

3. Get the CA certificate of the supervisor by running the following command against the cluster running the `pinniped-supervisor`:

```
kubectl get secret pinniped-supervisor-tls-cert -n pinniped-supervisor -o 'go-t
emplate={{index .data "tls.crt"}}'
```

> **Note**
>
> The `tls.crt` contains the entire certificate chain including the CA certificate for `letsencrypt` generated certificates.

4. Create the following resource to `workspace/pinniped-concierge/jwt_authenticator.yaml`:

```
---
apiVersion: authentication.concierge.pinniped.dev/v1alpha1
kind: JWTAuthenticator
metadata:
  name: pinniped-jwt-authenticator
spec:
  issuer: "DNS-NAME"
  audience: concierge
  tls:
    certificateAuthorityData: "CA-DATA"
```

If you use the `letsencrypt` production issuer, you can omit the `tls` section:

```
---
apiVersion: authentication.concierge.pinniped.dev/v1alpha1
kind: JWTAuthenticator
metadata:
  name: pinniped-jwt-authenticator
spec:
  issuer: "DNS-NAME"
  audience: concierge
```

Where:

- `DNS-NAME` is the domain in which the `pinniped-supervisor` is published. For example, `pinniped-supervisor.example.com`
- `CA-DATA` is the public key of the signing CA or the public key of the Pinniped httpproxy certificate.

5. Deploy the resource by running:

```
kapp deploy -y --app pinniped-concierge-jwt --into-ns pinniped-concierge -f pin
niped-concierge/jwt_authenticator.yaml
```

Log in to the cluster

See Log in by using Pinniped.

Integrate your Azure Active Directory

This topic tells you how to integrate your Azure Active Directory (commonly known as AD).

Integrate Azure AD with a new or existing AKS without Pinniped

Perform the following procedures to integrate Azure AD with a new or existing AKS without Pinniped.

Prerequisites

Meet these prerequisites:

- Download and install the Azure CLI
- Download and install the Tanzu CLI
- Download and install the Tanzu CLI RBAC plug-in

Set up a platform operator

To set up a platform operator:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **Groups** under the **Manage** side menu.
3. Identify or create an admin group for the AKS cluster.
4. Retrieve the object ID of the admin group.
5. Take one of the following actions.
    - Create an AKS Cluster with Azure AD enabled by running:

      ```
      az group create --name RESOURCE-GROUP --location LOCATION
      az aks create -g RESOURCE-GROUP -n MANAGED-CLUSTER --enable-aad --aad-adm
      in-group-object-ids OBJECT-ID
      ```

      Where:

        - `RESOURCE-GROUP` is your resource group
        - `LOCATION` is your location
        - `MANAGED-CLUSTER` is your managed cluster
        - `OBJECT-ID` is the object ID
    - Enable Azure AD integration on the existing cluster by running:

      ```
      az aks update -g RESOURCE-GROUP -n MANAGED-CLUSTER --enable-aad --aad-adm
      in-group-object-ids OBJECT-ID
      ```

      Where:

        - `RESOURCE-GROUP` is your resource group
        - `MANAGED-CLUSTER` is your managed cluster
        - `OBJECT-ID` is the object ID
6. Add **Platform Operators** to the admin group.
7. Log in to the AKS cluster by running:

    ```
    az aks get-credentials --resource-group RESOURCE-GROUP --name MANAGED-CLUSTER -
    -admin
    ```

    Where:

    - `RESOURCE-GROUP` is your resource group
    - `MANAGED-CLUSTER` is your managed cluster

Set up a Tanzu Application Platform default role group

To set up a Tanzu Application Platform default role group:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **Groups** under the **Manage** side menu.
3. Identify or create a list of groups in the Azure AD for each of the Tanzu Application Platform default roles (`app-operator`, `app-viewer`, and `app-editor`).
4. Retrieve the corresponding object IDs for each group.
5. Add users to the groups accordingly.

6. For each object ID retrieved earlier, use the Tanzu CLI RBAC plug-in to bind the `object id` group to a role by running:

   ```
   tanzu rbac binding add -g OBJECT-ID -r TAP-ROLE -n NAMESPACE
   ```

   Where:

   - `OBJECT-ID` is the object ID
   - `TAP-ROLE` is the Tanzu Application Platform role
   - `NAMESPACE` is the namespace

Set up kubeconfig

To set up kubeconfig:

1. Set up the `kubeconfig` to point to the AKS cluster by running:

   ```
   az aks get-credentials --resource-group RESOURCE-GROUP --name MANAGED-CLUSTER
   ```

   Where:

   - `RESOURCE-GROUP` is your resource group
   - `MANAGED-CLUSTER` is your managed cluster

2. Run any kubectl command to trigger a browser login. For example:

   ```
   kubectl get pods
   ```

Integrate Azure AD with Pinniped

Perform the following procedures to set up Azure AD with Pinniped.

Prerequisites

Meet these prerequisites:

- Download and install the Tanzu CLI
- Download and install the Tanzu CLI RBAC plug-in
- Install Pinniped supervisor and concierge on the cluster without setting up the OIDCIdentityProvider and secret.

Set up the Azure AD app

To set up the Azure AD app:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **App registrations** under the **Manage** side menu.
3. Select **New Registration**.
4. Enter the name of the application. For example, `gke-pinniped-supervisor-app`.
5. Under **Supported account types**, select **Accounts in this organisational directory only (VMware, Inc. only - Single tenant)**.
6. Under **Redirect URI**, select **Web** as the platform.
7. Enter the call URI to the supervisor. For example, `https://pinniped-supervisor.example.com/callback`.

8. Select **Register** to create the app.
9. If not already redirected, navigate to the app settings page.
10. Select **Token configuration** under the **Manage** menu.
11. Select **Add groups claim** > **All groups (includes distribution lists but not groups assigned to the application)**.
12. Select **Add** to create the group claim.
13. Select the app name in the breadcrumb navigation to return to the app settings page.
14. Select the **Endpoints** tab and record the value in the **OpenID Connect metadata document** field.
15. Return to the app settings page.
16. Record the **Application (client) ID**.
17. Select **Certificates & secrets** under the **Manage** menu.
18. Create a new client secret and record this value.
19. Add the following YAML to `oidc_identity_provider.yaml`.

```
---
apiVersion: idp.supervisor.pinniped.dev/v1alpha1
kind: OIDCIdentityProvider
metadata:
  namespace: pinniped-supervisor
  name: azure-ad
spec:
  # Specify the upstream issuer URL.
  issuer: ISSUER-URL

  authorizationConfig:
    additionalScopes: ["openid", "email", "profile"]
    allowPasswordGrant: false

  # Specify how claims are mapped to Kubernetes identities.
  claims:
    username: preferred_username
    groups: groups

  # Specify the name of the Kubernetes Secret that contains your
  # application's client credentials (created below).
  client:
    secretName: azure-ad-client-credentials
---
apiVersion: v1
kind: Secret
metadata:
  namespace: pinniped-supervisor
  name: azure-ad-client-credentials
type: secrets.pinniped.dev/oidc-client
stringData:
  clientID: "AZURE-AD-CLIENT-ID"
  clientSecret: "AZURE-AD-CLIENT-SECRET"
```

Where:

- `ISSUER-URL` is the OpenID Connect metadata document URL you recorded earlier, but without the trailing `/.well-known/openid-configuration`
- `AZURE-AD-CLIENT-ID` is the Azure AD client ID you recorded earlier
- `AZURE-AD-CLIENT-SECRET` is the Azure AD client secret you recorded earlier

20. Apply your changes from the kubectl CLI by running:

```
kubectl apply workspace/pinniped-supervisor/oidc_identity_provider.yaml
```

Set up the Tanzu Application Platform default role group

To set up a Tanzu Application Platform default role group:

1. Navigate to the **Azure Active Directory Overview** page.
2. Select **Groups** under the **Manage** side menu.
3. Identify or create a list of groups in the Azure AD for each of the Tanzu Application Platform default roles (`app-operator`, `app-viewer`, and `app-editor`).
4. Retrieve the corresponding object IDs for each group.
5. Add users to the groups accordingly.
6. For each object ID retrieved earlier, use the Tanzu CLI RBAC plug-in to bind the `object id` group to a role by running:

   ```
   tanzu rbac binding add -g OBJECT-ID -r TAP-ROLE -n NAMESPACE
   ```

   Where:

   - `OBJECT-ID` is the object ID
   - `TAP-ROLE` is the Tanzu Application Platform role
   - `NAMESPACE` is the namespace

Set up kubeconfig

Follow these steps to set up kubeconfig:

1. Set up `kubeconfig` using the Pinniped CLI by running:

   ```
   pinniped get kubeconfig --kubeconfig-context YOUR-KUBECONFIG-CONTEXT > /tmp/con
   cierge-kubeconfig
   ```

   Where `YOUR-KUBECONFIG-CONTEXT` is your your kubeconfig context.

2. Run any kubectl command to trigger a browser login. For example:

   ```
   export KUBECONFIG="/tmp/concierge-kubeconfig"
   kubectl get pods
   ```

Role descriptions for Tanzu Application Platform

This topic is a high level overview of each default role. For more information about the specific permissions of each role for every Tanzu Application Platform (commonly known as TAP) component, see Detailed role permissions for Tanzu Application Platform.

app-editor

The app-editor role can create, edit, and delete a Tanzu workload or deliverable.

Assign this role to a user, for example an app developer, to give permissions to create running workloads on the cluster. This allows them to deploy their applications. This role allows the user to:

- View, create, update, or delete a Tanzu workload or deliverable. This includes viewing the logs of the pods spun up through the Tanzu workload and tracing a commit through the build process.
- Download the images associated with their Tanzu workload so they can test images locally, or create a Tanzu workload from it instead of starting from source code in a repository.
- View and use Application Accelerator templates.
- View, create, update, or delete a Tanzu workload binding with an existing service.

app-viewer

The app-viewer role cannot create, edit, or delete a Tanzu workload or deliverable.

This role has a subset of the permissions of the app-editor role. Use it if you do not want a user to create, edit, or delete a Tanzu workload or deliverable, but they need to view its status. For example, give these permissions to an application developer that requires visibility into the state of their Tanzu workload or micro-service, but does not have the permissions to deploy it, such as to production or staging environments. This role cannot bind services with a Tanzu workload.

app-operator

The app-operator role can create, edit, and delete supply chain resources.

Assign this role to a user who defines the activities within a supply chain or the path to production. For example, building, testing, or scanning. This role can view, create, update, or delete Tanzu supply chain resources, including Tanzu Build Service control plane resources such as:

- kpack's builder, stack, and store
- Scanning resources
- Grype
- The metadata store

If this person must create Tanzu workloads, you can bind the user with the app-editor role as well.

service-operator

The service-operator role can create, edit, and delete service instances, service instance classes, and resource claim policies to permit the claimability of service instances across one or more namespaces.

Assign this role to a user who is responsible for the life cycle (create, edit and delete) of service instances. This role can also view resource claims across all namespaces as well as query for the list of claimable service instances in a given namespace.

workload

This role provides the service account associated with the Tanzu workload the permissions needed to execute the activities in the supply chain. This role is for a "robot" versus a user.

deliverable

This role gives the delivery "robot" service account the permissions needed to create running workloads. This role is not for a user.

Role descriptions for Tanzu Application Platform

This topic is a high level overview of each default role. For more information about the specific permissions of each role for every Tanzu Application Platform (commonly known as TAP) component, see Detailed role permissions for Tanzu Application Platform.

app-editor

The app-editor role can create, edit, and delete a Tanzu workload or deliverable.

Assign this role to a user, for example an app developer, to give permissions to create running workloads on the cluster. This allows them to deploy their applications. This role allows the user to:

- View, create, update, or delete a Tanzu workload or deliverable. This includes viewing the logs of the pods spun up through the Tanzu workload and tracing a commit through the build process.
- Download the images associated with their Tanzu workload so they can test images locally, or create a Tanzu workload from it instead of starting from source code in a repository.
- View and use Application Accelerator templates.
- View, create, update, or delete a Tanzu workload binding with an existing service.

app-viewer

The app-viewer role cannot create, edit, or delete a Tanzu workload or deliverable.

This role has a subset of the permissions of the app-editor role. Use it if you do not want a user to create, edit, or delete a Tanzu workload or deliverable, but they need to view its status. For example, give these permissions to an application developer that requires visibility into the state of their Tanzu workload or micro-service, but does not have the permissions to deploy it, such as to production or staging environments. This role cannot bind services with a Tanzu workload.

app-operator

The app-operator role can create, edit, and delete supply chain resources.

Assign this role to a user who defines the activities within a supply chain or the path to production. For example, building, testing, or scanning. This role can view, create, update, or delete Tanzu supply chain resources, including Tanzu Build Service control plane resources such as:

- kpack's builder, stack, and store
- Scanning resources
- Grype
- The metadata store

If this person must create Tanzu workloads, you can bind the user with the app-editor role as well.

service-operator

The service-operator role can create, edit, and delete service instances, service instance classes, and resource claim policies to permit the claimability of service instances across one or more namespaces.

Assign this role to a user who is responsible for the life cycle (create, edit and delete) of service instances. This role can also view resource claims across all namespaces as well as query for the list of claimable service instances in a given namespace.

workload

This role provides the service account associated with the Tanzu workload the permissions needed to execute the activities in the supply chain. This role is for a "robot" versus a user.

deliverable

This role gives the delivery “robot” service account the permissions needed to create running workloads. This role is not for a user.

Detailed role permissions for Tanzu Application Platform

This topic tells you the specific permissions of each role for every Tanzu Application Platform (commonly known as TAP) component.

Native Kubernetes Resources

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: [""]
  resources: ["configmaps","endpoints","events","persistentvolumeclaims","pods","pods/
log","resourcequotas","services"]
  verbs: ["get","list","watch"]
- apiGroups: ["apps"]
  resources: ["deployments","replicasets","statefulsets"]
  verbs: ["get","list","watch"]
- apiGroups: ["batch"]
  resources: ["cronjobs","jobs"]
  verbs: ["get","list","watch"]
- apiGroups: ["events.k8s.io"]
  resources: ["events"]
  verbs: ["get","list","watch"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator: "true"

```
- apiGroups: [""]
  resources: ["configmaps","secrets"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

App Accelerator

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: ["accelerator.apps.tanzu.vmware.com"]
  resources: ["accelerators"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator: "true"

```
- apiGroups: ["accelerator.apps.tanzu.vmware.com"]
  resources: ["accelerators"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Cartographer

apps.tanzu.vmware.com/aggregate-to-app-editor: "true"

```
- apiGroups: ["carto.run"]
  resources: ["deliverables","workloads"]
  verbs: ["create","patch","update","delete","deletecollection"]
```

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: ["carto.run"]
  resources: ["deliverables","runnables","workloads"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"

```
- apiGroups: ["carto.run"]
  resources: ["clusterconfigtemplates","clusterconfigtemplates","clusterdeliveries","c
lusterdeploymenttemplates","clusterimagetemplates","clusterruntemplates","clustersourc
etemplates","clustersupplychains","clustertemplates"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access

```
- apiGroups: ["carto.run"]
  resources: ["clusterconfigtemplates","clusterconfigtemplates","clusterdeliveries","c
lusterdeploymenttemplates","clusterimagetemplates","clusterruntemplates","clustersourc
etemplates","clustersupplychains","clustertemplates"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Cloud Native Runtimes

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: ["apps"]
  resources: ["deployments","replicasets","statefulsets"]
  verbs: ["get","list","watch"]
- apiGroups: ["batch"]
  resources: ["cronjobs","jobs"]
  verbs: ["get","list","watch"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get","list","watch"]
- apiGroups: ["eventing.knative.dev"]
  resources: ["brokers","triggers"]
  verbs: ["get","list","watch"]
- apiGroups: ["serving.knative.dev"]
  resources: ["configurations","services","revisions","routes"]
  verbs: ["get","list","watch"]
- apiGroups: ["sources.*"]
  resources: ["(many)"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator: "true"

```
- apiGroups: ["eventing.knative.dev"]
  resources: ["brokers"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["sources.*"]
  resources: ["(many)"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Convention Service

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: ["conventions.carto.run"]
  resources: ["podintents"]
  verbs: ["get","list","watch"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["podintents"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"

```
- apiGroups: ["conventions.carto.run"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access

```
- apiGroups: ["conventions.carto.run"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["clusterpodconventions"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Developer Conventions

apps.tanzu.vmware.com/aggregate-to-app-editor: "true"

```
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get","list","watch"]
- apiGroups: [""]
  resources: ["pods/exec","pods/portforward"]
  verbs: ["get","list","create"]
- apiGroups: ["carto.run"]
  resources: ["workloads"]
  verbs: ["get","list","watch"]
```

OOTB Templates

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get","list","watch"]
- apiGroups: ["carto.run"]
  resources: ["deliverables","runnables"]
  verbs: ["get","list","watch"]
- apiGroups: ["conventions.carto.run"]
  resources: ["podintents"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["podintents"]
  verbs: ["get","list","watch"]
- apiGroups: ["kappctrl.k14s.io"]
  resources: ["apps"]
  verbs: ["get","list","watch"]
- apiGroups: ["kpack.io"]
  resources: ["images"]
  verbs: ["get","list","watch"]
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["imagescans","sourcescans"]
  verbs: ["get","list","watch"]
- apiGroups: ["servicebinding.io"]
  resources: ["servicebindings"]
  verbs: ["get","list","watch"]
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch"]
- apiGroups: ["serving.knative.dev"]
  resources: ["services"]
  verbs: ["get","list","watch"]
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch"]
- apiGroups: ["source.toolkit.fluxcd.io"]
  resources: ["gitrepositories"]
  verbs: ["get","list","watch"]
- apiGroups: ["tekton.dev"]
  resources: ["pipelineruns","taskruns"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-workload: "true"

```
- apiGroups: ["carto.run"]
  resources: ["deliverables","runnables"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["conventions.carto.run"]
  resources: ["podintents"]
- apiGroups: ["conventions.apps.tanzu.vmware.com"]
  resources: ["podintents"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["kpack.io"]
  resources: ["images"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["imagescans","sourcescans"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.toolkit.fluxcd.io"]
  resources: ["gitrepositories"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["tekton.dev"]
  resources: ["pipelineruns","taskruns"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

`apps.tanzu.vmware.com/aggregate-to-deliverable: "true"`

```
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["kappctrl.k14s.io"]
  resources: ["apps"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["servicebinding.io"]
  resources: ["servicebindings"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["serving.knative.dev"]
  resources: ["services"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
- apiGroups: ["source.toolkit.fluxcd.io"]
  resources: ["gitrepositories"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Service Bindings

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["servicebinding.io"]
  resources: ["servicebindings"]
  verbs: ["get","list","watch"]
```

Services Toolkit

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"`

```
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["clusterresources"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["resourceclaims"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access`

```
- apiGroups: ["services.apps.tanzu.vmware.com"]
  resources: ["clusterresources"]
```

```
  verbs: ["get","list","watch"]
```

Source Controller

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["source.apps.tanzu.vmware.com"]
  resources: ["imagerepositories","mavenartifacts"]
  verbs: ["get","list","watch"]
```

Supply Chain Security Tools — Scan

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["imagescans","scanpolicies","scantemplates","sourcescans"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["scanning.apps.tanzu.vmware.com"]
  resources: ["scanpolicies","scantemplates"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Tanzu Build Service

`apps.tanzu.vmware.com/aggregate-to-app-editor: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["builds"]
  verbs: ["patch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["builds","builders","images"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["clusterbuilders","clusterstacks","clusterstores"]
  verbs: ["get","list","watch"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator: "true"`

```
- apiGroups: ["kpack.io"]
  resources: ["builders"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

`apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access`

```
- apiGroups: ["kpack.io"]
  resources: ["clusterbuilders","clusterstacks","clusterstores"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Tekton

apps.tanzu.vmware.com/aggregate-to-app-viewer: "true"

```
- apiGroups: ["tekton.dev"]
  resources: ["pipelineresources","pipelineruns","pipelines","taskruns","tasks"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-viewer-cluster-access: "true"

```
- apiGroups: ["tekton.dev"]
  resources: ["clustertasks"]
  verbs: ["get","list","watch"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator: "true"

```
- apiGroups: ["tekton.dev"]
  resources: ["pipelineresources","pipelines","tasks"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access

```
- apiGroups: ["tekton.dev"]
  resources: ["clustertasks"]
  verbs: ["get","list","watch","create","patch","update","delete","deletecollection"]
```

Bind a user or group to a default role

You can choose one of the following two approaches to bind a user or group to a default role:

- Use the Tanzu Application Platform RBAC CLI plug-in, which only supports binding Tanzu Application Platform (commonly known as TAP) default roles.
- Use Kubernetes role-based access control (RBAC) role binding.

VMware recommends that you use the Tanzu Application Platform RBAC CLI plug-in. This CLI plug-in simplifies the process by binding the cluster-scoped resource permissions at the same time as the namespace-scoped resource permissions, where applicable, for each default role. The following sections cover the Tanzu Application Platform RBAC CLI plug-in.

Prerequisites

1. Download the latest Tanzu CLI version.
2. Download the Tanzu Application Platform RBAC CLI plug-in `tar.gz` file from Tanzu Network.
3. Ensure you have admin access to the cluster.
4. Ensure you have configured an authentication solution for the cluster. You can use Pinniped or the authentication service native to your Kubernetes distribution.

Install the Tanzu Application Platform RBAC CLI plug-in

Follow these steps to install the Tanzu Application Platform RBAC CLI plug-in:

Caution

The Tanzu Application Platform RBAC CLI plug-in is currently in beta and is intended for evaluation and test purposes only.

1. Untar the `tar.gz` file:

   ```
   tar -zxvf NAME-OF-THE-TAR
   ```

2. Install the Tanzu Application Platform RBAC CLI plug-in locally on your operating system:

   macOS

   ```
   tanzu plugin install rbac --local darwin-amd64
   ```

   Linux

   ```
   tanzu plugin install rbac --local linux-amd64
   ```

   Windows

   ```
   tanzu plugin install rbac --local windows-amd64
   ```

(Optional) Use a different kubeconfig location

You can use a different kubeconfig location by running:

```
tanzu rbac --kubeconfig PATH-OF-KUBECONFIG binding add --user USER --role ROLE --names
pace NAMESPACE
```

Note

The environment variable `KUBECONFIG` is not implemented. You must use the `--kubeconfig` flag to enter a different location. Otherwise the default `~/.kube/config` is used.

For example:

```
$ tanzu rbac --kubeconfig /tmp/pinniped_kubeconfig.yaml binding add --user username@vm
ware.com --role app-editor --namespace user-ns
```

Add the specified user or group to a role

Add a user or group to a role by running:

```
tanzu rbac binding add --user USER --role ROLE --namespace NAMESPACE
```

```
tanzu rbac binding add --group GROUP --role ROLE --namespace NAMESPACE
```

For example:

```
$ tanzu rbac binding add --user username@vmware.com --role app-editor --namespace user
-ns
```

Get a list of users and groups from a role

Get a list of users and groups from a role by running:

```
tanzu rbac binding get --role ROLE --namespace NAMESPACE
```

For example:

```
$ tanzu rbac binding get --role app-editor --namespace user-ns
```

Remove the specified user or group from a role

Remove a user or group from a role by running:

```
tanzu rbac binding delete --user USER --role ROLE --namespace NAMESPACE
```

```
tanzu rbac binding delete --group GROUP --role ROLE --namespace NAMESPACE
```

For example:

```
$ tanzu rbac binding delete --user username@vmware.com --role app-editor --namespace u
ser-ns
```

Error logs

Authorization error logs might include the following errors:

- Permission Denied:

  The current user does not have permissions to create or edit rolebinding objects. Use an admin account when using the RBAC CLI.

  ```
  Error: rolebindings.rbac.authorization.k8s.io "app-operator" is forbidden: User
  "<subject>" cannot get resource "rolebindings" in API group "rbac.authorizatio
  n.k8s.io" in the namespace "namespace"
  Usage:
  tanzu rbac binding add [flags]
  Flags:
  -g, --group string User Group
  -h, --help help for add
  -n, --namespace string Namespace
  -r, --role string Role
  -u, --user string User Name

  Global Flags:
  --kubeconfig string kubeconfig file
  ```

- Already Bound Error:

  Adding a subject, user or group, to a role that already has the subject produces the following error:

```
Error: User ‘test-user’ is already bound to 'app-operator' role
Usage:
tanzu rbac binding add [flags]
Flags:
-g, --group string User Group
-h, --help help for add
-n, --namespace string Namespace
-r, --role string Role
-u, --user string User Name

Global Flags:
--kubeconfig string kubeconfig file
```

- Could Not Find Error:

  When removing a subject from a role, this error can occur in the following two scenarios:

  1. The rolebinding does not exist.
  2. The subject does not exist in the rolebinding.

  Ensure the rolebinding exists and that the subject name is correctly spelled.

```
Error: Did not find User 'test-user' in RoleBinding 'app-operator'
Usage:
tanzu rbac binding delete [flags]

Flags:
-g, --group string User Group
-h, --help help for delete
-n, --namespace string Namespace
-r, --role string Role
-u, --user string User Name

Global Flags:
--kubeconfig string kubeconfig file
```

- Object Has Been Modified Error:

  This error is a race condition caused by running multiple RBAC CLI actions at the same time. Rerunning the RBAC CLI might fix the issue.

```
Removed User 'test-user' from RoleBinding 'app-operator'
Removed User 'test-user' from ClusterRoleBinding 'app-operator-cluster-access'
Error: Operation cannot be fulfilled on rolebindings.rbac.authorization.k8s.io
"app-operator": the object has been modified; please apply your changes to the
latest version and try again
Usage:
tanzu rbac binding delete [flags]

Flags:
-g, --group string User Group
-h, --help help for delete
-n, --namespace string Namespace
-r, --role string Role
-u, --user string User Name
```

Troubleshooting

1. Get a list of permissions for a user or a group:

```
export NAME=SUBJECT-NAME
kubectl get rolebindings,clusterrolebindings -A -o json | jq -r ".items[] | sel
```

```
ect(.subjects[]?.name == \"${NAME}\") | .roleRef.name" | xargs -n1 kubectl desc
ribe clusterroles
```

2. Get a list of user or group for a specific role:

```
tanzu rbac binding get --role ROLE --namespace NAMESPACE
```

Log in to Tanzu Application Platform by using Pinniped

This topic tells you how to log in to your Tanzu Application Platform (commonly known as TAP) by using Pinniped.

As a prerequisite, the administrator must provide users access to resources by using `rolebindings`. It can be done with the `tanzu rbac` plug-in. For more information, see Bind a user or group to a default role.

To log in to your cluster by using Pinniped, follow these steps:

1. Install the Pinniped CLI.

   For more information, see Pinniped documentation.

   Important

   The latest compatible version of Pinniped CLI is required not only for the administrator to generate the `kubeconfig`, but also for the user to log in with the provided configuration.

2. Generate and distribute `kubeconfig` to users.
3. Login with the provided `kubeconfig`.

Download the Pinniped CLI

You must use a Pinniped CLI version that matches the installed Concierge or Supervisor. Use one of the following links to download the Pinniped CLI version `0.22.0`:

- Mac OS with AMD64
- Linux with AMD64
- Windows with AMD64

You must install the command-line tool on your `$PATH`, such as `/usr/local/bin` on macOS or Linux. You must also mark the file as executable.

Generate and distribute kubeconfig to users

As an administrator, you can generate the kubeconfig by using the following command:

```
pinniped get kubeconfig --kubeconfig-context <your-kubeconfig-context>  > /tmp/concier
ge-kubeconfig
```

Distribute this `kubeconfig` to your users so they can login by using `pinniped`.

Login with the provided kubeconfig

As a user of the cluster, you need the `kubeconfig` provided by your admin and the Pinniped CLI installed on your local machine to log in. Logging in is required to request information from the

cluster. You can execute any resource request with kubectl to enter the authentication flow. For example:

```
kubectl --kubeconfig /tmp/concierge-kubeconfig get pods
```

If you do not want to explicitly use `--kubeconfig` in every command, you can also export an environment variable to set the `kubeconfig` path in your shell session.

```
export KUBECONFIG="/tmp/concierge-kubeconfig"
kubectl get pods
```

This command enables `pinniped` to print a URL for you to visit in the browser. You can then log in, copy the authentication code and paste it back to the terminal. After the login succeeds, you either see the resources or a message indicating that you have no permission to access the resources.

If you use a Windows machine, the command referenced in the generated `kubeconfig` might not work. In this case, you must change the path under `user.exec.command` in the `kubeconfig` to point to the install path of the Pinniped CLI.

Additional resources about Tanzu Application Platform authentication and authorization

Use this topic to learn additional information about authentication and authorization for Tanzu Application Platform (commonly known as TAP).

See Default roles for Tanzu Application Platform overview to get started.

Install

Default roles are released as part of Tanzu Application Platform. Alternatively, you can also install default roles independently. See Install default roles independently for more information.

Note

The `tanzu rbac` CLI plug-in requires a separate installation.

Additional resources about Tanzu Application Platform authentication and authorization

Use this topic to learn additional information about authentication and authorization for Tanzu Application Platform (commonly known as TAP).

See Default roles for Tanzu Application Platform overview to get started.

Install

Default roles are released as part of Tanzu Application Platform. Alternatively, you can also install default roles independently. See Install default roles independently for more information.

Note

The `tanzu rbac` CLI plug-in requires a separate installation.

Install default roles independently for your Tanzu Application Platform

This topic tells you how to install default roles for Tanzu Application Platform (commonly known as TAP) without deploying a TAP profile.

Note

Follow the steps in this topic if you do not want to use a profile to install default roles. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing default roles, complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install default roles:

1. List version information for the package by running:

   ```
   tanzu package available list tap-auth.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list tap-auth.tanzu.vmware.com --namespace tap-instal
   l
   - Retrieving package versions for tap-auth.tanzu.vmware.com...
     NAME                          VERSION       RELEASED-AT
     tap-auth.tanzu.vmware.com     1.0.1
   ```

2. Install the package by running:

   ```
   tanzu package install tap-auth \
     --package tap-auth.tanzu.vmware.com \
     --version VERSION \
     --namespace tap-install
   ```

   Where:

   - `VERSION` is the package version number. For example, `1.0.1`.

   For example:

   ```
   $ tanzu package install tap-auth \
     --package tap-auth.tanzu.vmware.com \
     --version 1.0.1 \
     --namespace tap-install
   ```

Overview of Bitnami Services

Bitnami Services provides a of backing services for Tanzu Application Platform (commonly known as TAP). The services are MySQL, PostgreSQL, RabbitMQ, and Redis, all of which are backed by the corresponding Bitnami Helm Chart.

Through integration with Crossplane and Services Toolkit, these four services are immediately ready for apps teams to consume, with no additional setup or configuration required from ops teams. This makes it incredibly quick and easy to get started working with services on Tanzu Application Platform.

Note

The Bitnami Services package provides unmanaged services that are not designed to support long lived instances. Therefore, there is no supported path to upgrade individual instances. Bitnami Services are instantiated using the configuration and version of the package at creation time and so upgrading the Bitnami Services package has no effect on existing instances. VMware discourages changing the `compositionUpdatePolicy` and `compositionRevisionRef` on the individual composite resources (XRs) because this might cause unintended side effects.

Getting started

If this is your first time working with Bitnami Services on Tanzu Application Platform, you can start with the tutorial Working with Bitnami Services. Otherwise, see the how-to guides and reference material.

Overview of Bitnami Services

Bitnami Services provides a of backing services for Tanzu Application Platform (commonly known as TAP). The services are MySQL, PostgreSQL, RabbitMQ, and Redis, all of which are backed by the corresponding Bitnami Helm Chart.

Through integration with Crossplane and Services Toolkit, these four services are immediately ready for apps teams to consume, with no additional setup or configuration required from ops teams. This makes it incredibly quick and easy to get started working with services on Tanzu Application Platform.

Note

The Bitnami Services package provides unmanaged services that are not designed to support long lived instances. Therefore, there is no supported path to upgrade individual instances. Bitnami Services are instantiated using the configuration and version of the package at creation time and so upgrading the Bitnami Services package has no effect on existing instances. VMware discourages changing the `compositionUpdatePolicy` and `compositionRevisionRef` on the individual composite resources (XRs) because this might cause unintended side effects.

Getting started

If this is your first time working with Bitnami Services on Tanzu Application Platform, you can start with the tutorial Working with Bitnami Services. Otherwise, see the how-to guides and reference material.

Install Bitnami Services

This topic tells you how to install Bitnami Services from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Bitnami Services. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Bitnami Services, you must:

- Fulfill all prerequisites for installing Tanzu Application Platform
- Install Crossplane
- Install Services Toolkit

Install Bitnami Services

To install Bitnami Services:

1. See what versions of Bitnami Services are available to install by running:

   ```
   tanzu package available list -n tap-install bitnami.services.tanzu.vmware.com
   ```

   For example:

   ```
   $ tanzu package available list -n tap-install bitnami.services.tanzu.vmware.com
     NAME                               VERSION           RELEASED-AT
     bitnami.services.tanzu.vmware.com  0.1.0             2023-03-10 14:35:15 +000
   0 UTC
   ```

2. Install Bitnami Services by running:

   ```
   tanzu package install bitnami-services \
     --package bitnami.services.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the Bitnami Services version you want to install. For example, `0.1.0`.

3. Verify that the package installed by running:

   ```
   tanzu package installed get bitnami-services -n tap-install
   ```

   In the output, confirm that the `STATUS` value is `Reconcile succeeded`.

   For example:

   ```
   $ tanzu package installed get bitnami-services -n tap-install
   NAMESPACE:          tap-install
   NAME:               bitnami-services
   PACKAGE-NAME:       bitnami.services.tanzu.vmware.com
   PACKAGE-VERSION:    0.1.0
   STATUS:             Reconcile succeeded
   CONDITIONS:         - type: ReconcileSucceeded
     status: "True"
     reason: ""
     message: ""
   ```

Bitnami Services tutorials

This section contains tutorials for how to use Bitnami Services.

In this section:

- Working with Bitnami Services

Working with Bitnami Services

In this tutorial you learn how application operators can discover, claim, and bind services to application workloads.

Tanzu Application Platform has four services that are available in the Bitnami Services package. These are MySQL, PostgreSQL, RabbitMQ, and Redis. The corresponding Bitnami Helm Chart backs each of these services.

About this tutorial

Target user role: Application Operator
Complexity: Basic
Estimated time: 15 minutes
Topics covered: Classes, Claims, Bitnami,
Learning outcomes: An understanding of how work with the standard Bitnami services

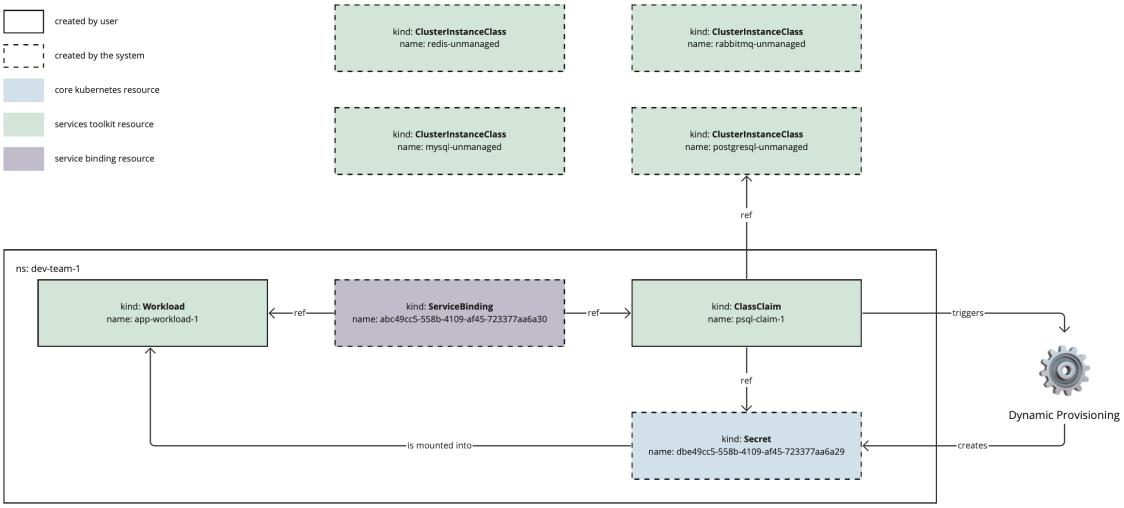
Prerequisites

To follow this tutorial, you must have:

- Access to a Tanzu Application Platform cluster v1.5.0 and later
- The Tanzu services CLI plug-in v0.6.0 and later

Concepts

The following diagram provides an overview of the elements you will use during this tutorial and how they all fit together.



In this diagram:

- There are only two elements that require user input, which are creating a `ClassClaim` and creating a `Workload`. The workload is configured to refer to the class claim`.

- The life cycles of the `ClassClaim` and the `Workload` are separate. This allows you to update one without affecting the other.
- The dynamic provisioning process is simplified. This is intentional because Application Operators and Developers do not need to know about the inner workings and configurations of service instances.

Procedure

The following steps explain how to work with Bitnami Services.

Step 1: Discover services

Application teams can discover the range of services on offer to them by running:

```
tanzu service class list
```

The expected output is similar to the following:

```
  NAME                  DESCRIPTION
  mysql-unmanaged       MySQL by Bitnami
  postgresql-unmanaged  PostgreSQL by Bitnami
  rabbitmq-unmanaged    RabbitMQ by Bitnami
  redis-unmanaged       Redis by Bitnami
```

Here the output shows four classes. These are the four Bitnami Services available in the Bitnami services package. You can see from the names and descriptions that they are all *unmanaged* services. This implies that the resulting service instances run on cluster, that is, they are not a managed service running in the cloud. Other classes might be listed here as well.

As an Application Operator, you review the classes on offer and choose one that meets your requirements.

You can learn and discover more about a class by running:

```
tanzu service class get postgresql-unmanaged
```

Example output:

```
NAME:           postgresql-unmanaged
DESCRIPTION:    PostgreSQL by Bitnami
READY:          true

PARAMETERS:
  KEY        DESCRIPTION                                                TYPE     DEF
AULT  REQUIRED
  storageGB  The desired storage capacity of the database, in Gigabytes.  integer  1
false
```

The output shows the name and a short description for the class, its current status, and the parameters. The parameters represent the set of configuration options that are available to application teams.

The `postgresql-unmanaged` class here has one parameter, which is `storageGB`. You can also see that it is not required to pass this parameter when creating a claim for the class, in which case the default value of `1` is used.

Step 2: Claim services

In this example, you have an application workload that requires a PostgreSQL database to function correctly. You can claim the PostgreSQL Bitnami Service to obtain such a database.

To create the claim in a namespace named `dev-team-1`, you must first create the namespace by running:

```
kubectl create namespace dev-team-1
```

You can use the `tanzu service class-claim create` command to create a claim for the `postgresql-unmanaged` class, then bind your application workload to the resulting claim. In this example, you are also choosing to override the default value of `1` for the `storageGB` parameter, setting it instead to `3`. You can override any of the options as you see fit.

```
tanzu service class-claim create psql-1 --class postgresql-unmanaged --parameter stora
geGB=3 -n dev-team-1
```

Example output:

```
Creating claim 'psql-1' in namespace 'dev-team-1'.
Please run `tanzu service class-claim get psql-1 --namespace dev-team-1` to see the pr
ogress of create.
```

As the output states, you can then confirm the status of the claim by using the `tanzu service class-claim get` command as follows:

```
tanzu service class-claim get psql-1 --namespace dev-team-1
```

Example output:

```
Name: psql-1
Namespace: dev-team-1
Claim Reference: services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:psql-1
Class Reference:
  Name: postgresql-unmanaged
Parameters:
  storageGB: 3
Status:
  Ready: True
  Claimed Resource:
    Name: 7974379c-7b4d-41c3-af57-f4f1ae08c65d
    Namespace: dev-team-1
    Group:
    Version: v1
    Kind: Secret
```

It might take a moment or two before the claim reports `Ready: True`. After the claim is ready, you then have a successful claim for a PostgreSQL database configured to your needs with 3 GB of storage.

Step 3: Bind the claim to a workload

After creating the claim, you can bind it to one or more of your application workloads.

Important

If binding to more than one application workload then all application workloads must exist in the same namespace. This is a known limitation. For more information, see

> Cannot claim and bind to the same service instance from across multiple namespaces.

1. Find the reference for the claim by running the following command.

   ```
   tanzu service class-claim get psql-1
   ```

   The reference is in the output under the heading Claim Reference.

2. Bind the claim to a workload of your choice by pass a reference to the claim to the `--service-ref` flag of the `tanzu apps workload create` command. For example:

   ```
   tanzu apps workload create my-workload --image my-registry/my-app-image --servi
   ce-ref db=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:psql-1
   ```

   You must pass the claim reference with a corresponding name that follows the format `--service-ref db=services.apps.tanzu.vmware.com/v1alpha1:ClassClaim:psql-1`. The `db=` prefix to this example reference is an arbitrary name for the reference.

Bitnami Services how-to guides

This section contains how-to guides for Bitnami Services.

In this section:

- Configure private registry and VMware Tanzu Application Catalog integration for Bitnami Services
- Obtain credentials for VMware Tanzu Application Catalog integration
- Troubleshooting and known limitations

Configure private registry and VMware Tanzu Application Catalog integration for Bitnami Services

This topic tells you how to integrate Bitnami Services with private registries or with VMware Tanzu Application Catalog. You can configure this globally for all services, or on a per-service basis.

Prerequisites

Before you integrate Bitnami Services with a private registry or VMware Tanzu Application Catalog, you must:

- Have your Helm Chart repository URL in the format `oci://REGISTRY-HOSTNAME/REPOSITORY-PATH/charts`.
- Have the credentials to access the private registry.

For how to obtain both of these prerequisites for VMware Tanzu Application Catalog integration, see Obtain credentials for VMware Tanzu Application Catalog integration.

Procedure

1. Create two Kubernetes `Secrets`, one with credentials to pull Helm charts and the other with credentials to pull images. The following examples put these in the `default` namespace, but you can choose to place them in any namespace you prefer.

```
$ kubectl create secret generic tac-chart-pull \
  -n default \
  --from-literal=username='USERNAME' \
  --from-literal=password='TOKEN'
```

```
$ kubectl create secret docker-registry tac-container-pull \
  -n default \
  --docker-server='REGISTRY-HOSTNAME' \
  --docker-username='USERNAME' \
  --docker-password='TOKEN'
```

2. Apply the configuration either to all Bitnami services or to one specific service.
    - **Apply configuration to all Bitnami services:**
        1. Add the following to your `tap-values.yaml` file:

```
bitnami_services:
  globals:
    helm_chart:
      repo: oci://REGISTRY-HOSTNAME/REPOSITORY-PATH/charts # Updat
e this value.
      chart_pull_secret_ref:
        name: tac-chart-pull
        namespace: default
      container_pull_secret_ref:
        name: tac-container-pull
        namespace: default
```

        2. Update Tanzu Application Platform by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com --value
s-file tap-values.yaml -n tap-install
```

    - **Apply configuration to one specific Bitnami service:**
        1. Add the following to your `tap-values.yaml` file:

```
bitnami_services:
  mysql: # choose from 'mysql', 'postgresql', 'rabbitmq' and 'redi
s'
    helm_chart:
      repo: oci://REGISTRY-HOSTNAME/REPOSITORY-PATH/charts # Updat
e this value.
      chart_pull_secret_ref:
        name: tac-chart-pull
        namespace: default
      container_pull_secret_ref:
        name: tac-container-pull
        namespace: default
```

        2. Update Tanzu Application Platform by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com --value
s-file tap-values.yaml -n tap-install
```

Known issue

As of Tanzu Application Platform v1.5.0 there is a known issue that occurs if you try to configure private registry integration for the Bitnami services after having already created a claim for one or more of the Bitnami services using the default configuration. The issue is that the updated private

registry configuration does not appear to take effect. This is due to caching behavior in the system which is not currently accounted for during configuration updates.

Workaround

There is a temporary workaround to this issue, which is to delete the `provider-helm-*` pods in the `crossplane-system` namespace and wait for new pods to come back online after having applied updated registry configuration.

Obtain credentials for VMware Tanzu Application Catalog integration with Bitnami Services

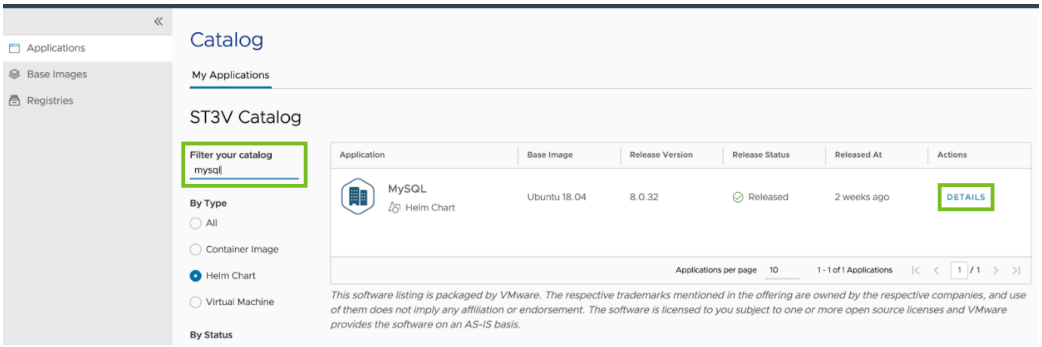
This topic tells you how to obtain credentials for VMware Tanzu Application Catalog to use when following the procedure in Configure private registry and VMware Tanzu Application Catalog integration for Bitnami Services.

Prerequisites

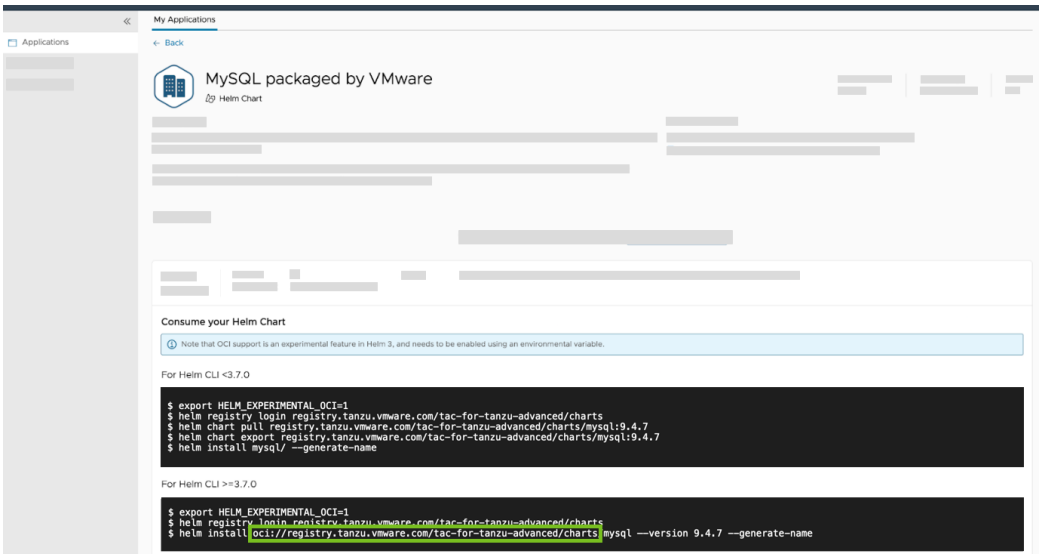
Before obtaining credentials, you must have a VMware Tanzu Application Catalog instance that can create access tokens from within the VMware Tanzu Application Catalog UI.

Obtain the Helm chart repository for VMware Tanzu Application Catalog

1. In VMware Tanzu Application Catalog, navigate to the **Applications** side tab.
2. Under **Filter your catalog**, search for Helm Charts in your catalog, for example, `MySQL`, and click **Details** for one of the charts you found:

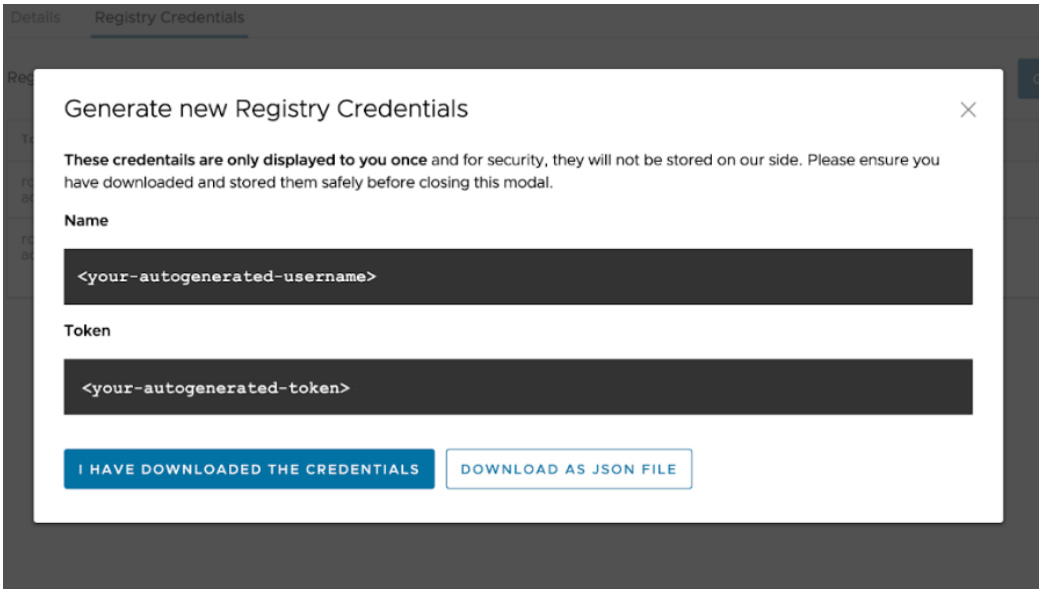


3. Take note of the repository shown under **For Helm CLI >= 3.7.0**. You must include the `oci://` prefix as shown on the page:



Obtain pull credentials for VMware Tanzu Application Catalog

1. In VMware Tanzu Application Catalog, navigate to the **Registries** side tab:
2. Click on the registry that contains your Helm Charts and container images and record the **Registry URL**.
3. Click the **Registry Credentials** tab.
4. Click **Generate New Credentials**.
5. Record the user name and token you are presented with.



You can now take the repository, user name, and token and use it to configure VMware Tanzu Application Catalog integration with the Bitnami services by following the steps in Configure Private Registry and VMware Tanzu Application Catalog Integration for Bitnami Services.

Troubleshoot Bitnami Services

This topic explains how you troubleshoot issues related to Bitnami Services on Tanzu Application Platform (commonly known as TAP).

Private registry or VMware Tanzu Application Catalog configuration does not take effect

Symptom:

If you configure private registry integration for the Bitnami services after creating a claim for a Bitnami service using the default configuration, the updated private registry configuration does not appear to take effect.

Cause:

This is due to caching behavior in the system that is not accounted for during configuration updates.

Solution:

Delete the `provider-helm-*` pods in the `crossplane-system` namespace and wait for new pods to come back online after having applied the updated registry configuration.

Bitnami Services reference

This section provides reference documentation for Bitnami Services.

In this section:

- Dependencies
- Package values

Dependencies for Bitnami Services

Bitnami Services is an integration package, which means that it provides configuration for other Tanzu Application Platform (commonly known as TAP) components. As such, it has a number of dependencies that must be met before you can use it.

The dependencies for Bitnami Services are:

- **Crossplane and the two providers** `provider-helm` **and** `provider-kubernetes`: See Install Crossplane
- **Services Toolkit**: See Install Services Toolkit

These dependencies are met if you install Tanzu Application Platform using the full, iterate, or run profiles.

Package values for Bitnami Services

This topic lists the keys and values that you can use to configure the behavior of the Bitnami Services package. You can apply configuration globally to all services using the `globals` key, or on a per-service basis using the `mysql`, `postgresql`, `rabbitmq` and `redis` keys.

If you are applying configuration to Tanzu Application Platform through the use of profiles and the `tap-values.yaml`, all configuration must exist under the `bitnami_services` top-level key.

For example:

```
bitnami_services:
  globals:
    helm_chart:
      # If you choose to use a custom Helm Chart repo, it's possible you'll also need
to configure specific versions
      # for each Chart as well, see example configuration below for postgresql.
```

```
    repo: https://charts.mycompany.example.com
mysql:
  enabled: false
postgresql:
  helm_chart:
    version: 12.2.6
  instance_class:
    name: company-redis
    description: My company postgres
rabbitmq:
  instance_class:
    name: company-redis
    description: My company rabbit
redis:
  instance_class:
    name: company-redis
    description: My company redis
```

Globals

The following table lists configuration that applies to all services.

KEY	DEFAULT	TYPE	DESCRIPTION
globals.create_clusterroles	`true`	boolean	Optional: Specifies whether to create default ClusterRoles that grant `claim` permissions to all Tanzu Application Platform Application operators.
globals.helm_chart.chart_pull_secret_ref.name	`""`	string	Name of the pull secret. Can be overridden by individual services.
globals.helm_chart.chart_pull_secret_ref.namespace	`""`	string	Namespace of the pull secret. Can be overridden by individual services.
globals.helm_chart.container_pull_secret_ref.name	`""`	string	Name of the secret. Can be overridden by individual services.
globals.helm_chart.container_pull_secret_ref.namespace	`""`	string	Namespace of the secret. Can be overridden by individual services.
globals.helm_chart.repo	`https://charts.bitnami.com/bitnami`	string	Optional: Repository hosting the Helm charts used to provision the instances of all services. Can be overridden by individual services.
globals.shared_namespace	`""`	string	Optional: Name of the namespace that is shared by all provisioned instances of all services. By default, each instance is provisioned in its own dedicated namespace. Can be overridden by individual services.

MySQL

The following table lists configuration that applies to the `mysql` service.

KEY	DEFAULT	TYPE	DESCRIPTION
mysql.defaults.storage_size_gb	`1`	integer	Optional: The amount of storage to give each MySQL instance by default, in Gigabytes.
mysql.enabled	`true`	boolean	Optional: Provide developers an offering for unmanaged MySQL instances.

KEY	DEFAULT	TYPE	DESCRIPTION
mysql.helm_chart.repo	""	string	Optional: Repository hosting the Helm chart used to provision MySQL instances.
mysql.helm_chart.version	9.5.0	string	Optional: Version of the Helm chart used to provision MySQL instances.
mysql.helm_chart.chart_pull_secret_ref.name	""	string	Name of the pull secret.
mysql.helm_chart.chart_pull_secret_ref.namespace	""	string	Namespace of the pull secret.
mysql.helm_chart.container_pull_secret_ref.name	""	string	Name of the secret. Can be overridden by individual services.
mysql.helm_chart.container_pull_secret_ref.namespace	""	string	Namespace of the secret. Can be overridden by individual services.
mysql.instance_class.description	MySQL by Bitnami	string	Optional: Description of the ClusterInstanceClass that developers use to provision and claim MySQL instances.
mysql.instance_class.name	mysql-unmanaged	string	Optional: Name of the ClusterInstanceClass that developers use to provision and claim MySQL instances.
mysql.shared_namespace	""	string	Optional: Name of the namespace that is shared by all provisioned MySQL instances. By default, each instance is provisioned in its own dedicated namespace.

PostgreSQL

The following table lists configuration that applies to the `postgresql` service.

KEY	DEFAULT	TYPE	DESCRIPTION
postgresql.enabled	true	boolean	Optional: Provide developers an offering for unmanaged PostgreSQL instances.
postgresql.helm_chart.chart_pull_secret_ref.name	""	string	Name of the pull secret.
postgresql.helm_chart.chart_pull_secret_ref.namespace	""	string	Namespace of the pull secret.
postgresql.helm_chart.container_pull_secret_ref.name	""	string	Name of the secret. Can be overridden by individual services.
postgresql.helm_chart.container_pull_secret_ref.namespace	""	string	Namespace of the secret. Can be overridden by individual services.
postgresql.helm_chart.repo	""	string	Optional: Repository hosting the Helm chart used to provision PostgreSQL instances.
postgresql.helm_chart.version	12.2.0	string	Optional: Version of the Helm chart used to provision PostgreSQL instances.
postgresql.instance_class.description	PostgreSQL by Bitnami	string	Optional: Description of the ClusterInstanceClass that developers use to provision and claim PostgreSQL instances.

KEY	DEFAULT	TYPE	DESCRIPTION
postgresql.instance_class.name	`postgresql-unmanaged`	string	Optional: Name of the ClusterInstanceClass that developers use to provision and claim PostgreSQL instances.
postgresql.shared_namespace	`""`	string	Optional: Name of the namespace that is shared by all provisioned PostgreSQL instances. By default, each instance will be provisioned in its own dedicated namespace.
postgresql.defaults.storage_size_gb	`1`	integer	Optional: The amount of storage to give each PostgreSQL instance by default, in Gigabytes.

RabbitMQ

The following table lists configuration that applies to the `rabbitmq` service.

KEY	DEFAULT	TYPE	DESCRIPTION
rabbitmq.enabled	`true`	boolean	Optional: Provide developers an offering for unmanaged RabbitMQ instances
rabbitmq.helm_chart.contaier_pull_secret_ref.name	`""`	string	Name of the secret. Can be overridden by individual services.
rabbitmq.helm_chart.contaier_pull_secret_ref.namespace	`""`	string	Namespace of the secret. Can be overridden by individual services.
rabbitmq.helm_chart.repo	`""`	string	Optional: Repository hosting the Helm chart used to provision RabbitMQ instances.
rabbitmq.helm_chart.version	`11.10.0`	string	Optional: Version of the Helm chart used to provision RabbitMQ instances.
rabbitmq.helm_chart.chart_pull_secret_ref.name	`""`	string	Name of the pull secret.
rabbitmq.helm_chart.chart_pull_secret_ref.namespace	`""`	string	Namespace of the pull secret.
rabbitmq.instance_class.description	`RabbitMQ by Bitnami`	string	Optional: Description of the ClusterInstanceClass that developers use to provision and claim RabbitMQ instances.
rabbitmq.instance_class.name	`rabbitmq-unmanaged`	string	Optional: Name of the ClusterInstanceClass that developers use to provision and claim RabbitMQ instances.
rabbitmq.shared_namespace	`""`	string	Optional: Name of the namespace that is shared by all provisioned RabbitMQ instances. By default, each instance will be provisioned in its own dedicated namespace.
rabbitmq.defaults.replica_count	`1`	integer	Optional: The number of replicas to create for each RabbitMQ instance by default.
rabbitmq.defaults.storage_size_gb	`1`	integer	Optional: The amount of storage to give each RabbitMQ instance by default, in Gigabytes.

Redis

The following table lists configuration that applies to the `redis` service.

KEY	DEFAULT	TYPE	DESCRIPTION
redis.instance_class.description	`Redis by Bitnami`	string	Optional: Description of the ClusterInstanceClass that is used by developers to provision and claim Redis instances.
redis.instance_class.name	`redis-unmanaged`	string	Optional: Name of the ClusterInstanceClass that is used by developers to provision and claim Redis instances.
redis.shared_namespace	`""`	string	Optional: Name of the namespace that is shared by all provisioned Redis instances. By default, each instance will be provisioned in its own dedicated namespace.
redis.defaults.storage_size_gb	`1`	integer	Optional: The amount of storage to give each Redis instance by default, in Gigabytes.
redis.enabled	`true`	boolean	Optional: Provide developers an offering for unmanaged Redis instances.
redis.helm_chart.chart_pull_secret_ref.name	`""`	string	Name of the pull secret.
redis.helm_chart.chart_pull_secret_ref.namespace	`""`	string	Namespace of the pull secret.
redis.helm_chart.container_pull_secret_ref.name	`""`	string	Name of the secret. Can be overridden by individual services.
redis.helm_chart.container_pull_secret_ref.namespace	`""`	string	Namespace of the secret. Can be overridden by individual services.
redis.helm_chart.repo	`""`	string	Optional: Repository hosting the Helm chart used to provision Redis instances.
redis.helm_chart.version	`17.8.0`	string	Optional: Version of the Helm chart used to provision Redis instances.

Version matrix for Bitnami Services

This topic provides you with a version matrix for the Bitnami Services package and its open source components in Tanzu Application Platform v1.5 (commonly known as TAP).

To view this information for another Tanzu Application Platform version, select the version from the drop-down menu at the top of this page.

The following table has the component versions for the Bitnami Services package.

Component	Version
Bitnami Services package	0.1.0
MySQL Chart	9.5.0
PostgreSQL Chart	12.2.0
RabbitMQ Chart	11.10.0
Redis Chart	17.8.0

Note

Tanzu Application Platform patch releases are only added to the table when there is a change to one or more of the other versions in the table. Otherwise, the

corresponding versions remain the same for each Tanzu Application Platform patch release.

Overview of Cartographer Conventions

This topic describes an overview of Cartographer Conventions and how you can use it with Tanzu Application Platform.

Overview

Cartographer Conventions provides a means for operators to express their knowledge about how applications can run on Kubernetes as a convention. Cartographer Conventions supports defining and applying conventions to pods. It applies these opinions to fleets of developer workloads as they are deployed to the platform, saving operator and developer time.

The service is composed of two components

- **convention controller:** The convention service's convention controller provides the metadata to the convention server and executes the updates to a PodTemplateSpec in accordance with convention server's requests.
- **convention server:** The convention server receives and evaluates metadata associated with a workload and requests updates to the PodTemplateSpec associated with that workload. You can have one or more convention servers for a single controller instance.

About applying conventions

The convention server uses criteria defined in the convention to discover whether the configuration of a workload must change. The server receives the OCI metadata from the convention controller. If the metadata meets the criteria defined by the convention server, the conventions are applied. A convention can apply to all workloads regardless of metadata.

Applying conventions by using image metadata

You can define conventions to target workloads by using properties of their OCI metadata.

Conventions can use this information to only apply changes to the configuration of workloads when they match specific criteria. Such as, Spring Boot or .Net apps, or Spring Boot v2.3+. Targeted conventions can ensure that uniformity across specific workload types deployed on the cluster.

You can use all the metadata details of an image when evaluating workloads. To see the metadata details, use the Docker CLI command:

```
docker image inspect IMAGE`.
```

Note

Depending on how the image was built, metadata might not be available to reliably identify the image type and match the criteria for a convention server. Images built with Cloud Native Buildpacks reliably include rich descriptive metadata. Images built by some other process might not include the same metadata.

Applying conventions without using image metadata

Conventions can apply to workloads without targeting build service metadata. Examples of possible uses of this type of convention include:

- appending a logging or metrics sidecar,
- adding environment variables, or
- adding cached volumes.

These kinds of conventions ensure that infrastructure uniformity exists across workloads deployed on the cluster while reducing developer toil.

Important

Adding a sidecar alone does not make the log or metrics collection work. This requires having collector agents deployed and accessible from the Kubernetes cluster, and configuring required access by using role-based access control (RBAC) policy.

Overview of Cartographer Conventions

This topic describes an overview of Cartographer Conventions and how you can use it with Tanzu Application Platform.

Overview

Cartographer Conventions provides a means for operators to express their knowledge about how applications can run on Kubernetes as a convention. Cartographer Conventions supports defining and applying conventions to pods. It applies these opinions to fleets of developer workloads as they are deployed to the platform, saving operator and developer time.

The service is composed of two components

- **convention controller:** The convention service's convention controller provides the metadata to the convention server and executes the updates to a PodTemplateSpec in accordance with convention server's requests.
- **convention server:** The convention server receives and evaluates metadata associated with a workload and requests updates to the PodTemplateSpec associated with that workload. You can have one or more convention servers for a single controller instance.

About applying conventions

The convention server uses criteria defined in the convention to discover whether the configuration of a workload must change. The server receives the OCI metadata from the convention controller. If the metadata meets the criteria defined by the convention server, the conventions are applied. A convention can apply to all workloads regardless of metadata.

Applying conventions by using image metadata

You can define conventions to target workloads by using properties of their OCI metadata.

Conventions can use this information to only apply changes to the configuration of workloads when they match specific criteria. Such as, Spring Boot or .Net apps, or Spring Boot v2.3+. Targeted conventions can ensure that uniformity across specific workload types deployed on the cluster.

You can use all the metadata details of an image when evaluating workloads. To see the metadata details, use the Docker CLI command:

```
docker image inspect IMAGE`.
```

Note

Depending on how the image was built, metadata might not be available to reliably identify the image type and match the criteria for a convention server. Images built with Cloud Native Buildpacks reliably include rich descriptive metadata. Images built by some other process might not include the same metadata.

Applying conventions without using image metadata

Conventions can apply to workloads without targeting build service metadata. Examples of possible uses of this type of convention include:

- appending a logging or metrics sidecar,
- adding environment variables, or
- adding cached volumes.

These kinds of conventions ensure that infrastructure uniformity exists across workloads deployed on the cluster while reducing developer toil.

Important

Adding a sidecar alone does not make the log or metrics collection work. This requires having collector agents deployed and accessible from the Kubernetes cluster, and configuring required access by using role-based access control (RBAC) policy.

Install Cartographer Conventions

Cartographer Conventions is bundled with Supply Chain Choreographer as of the v0.4.0 release. See Installing Supply Chain Choreographer.

Create conventions with Cartographer Conventions

This topic describes how you can create and deploy custom conventions to the Tanzu Application Platform by using Cartographer Conventions.

Introduction

Tanzu Application Platform helps developers transform their code into containerized workloads with a URL. The Supply Chain Choreographer for Tanzu manages this transformation. For more information, see Supply Chain Choreographer.

Cartographer Conventions is a key component of the supply chain compositions the choreographer calls into action. Cartographer Conventions enables people in operational roles to efficiently apply their expertise. They can specify the runtime best practices, policies, and conventions of their organization to workloads as they are created on the platform. The power of this component becomes evident when the conventions of an organization are applied consistently, at scale, and without hindering the velocity of application developers.

Opinions and policies vary from organization to organization. Cartographer Convention supports the creation of custom conventions to meet the unique operational needs and requirements of an

organization.

Before jumping into the details of creating a custom convention, you can view two distinct components of Cartographer Conventions:

- Convention controller
- Convention server

Convention server

The convention server is the component that applies a convention already defined on the server. For a golang example of creating a convention server to add Spring Boot conventions, see spring-convention-server in GitHub. The resource that structures the request body of the request and response from the server is the PodConventionContext.

The `PodConventionContext` is a webhooks.conventions.carto.run/v1alpha1 type that defines the structure used to communicate internally by the webhook convention server. It ***does not exist*** on the Kubernetes API Server.

`PodConventionContext` is a wrapper for two types:

- `PodConventionContextSpec` which acts as a wrapper for a `PodTemplateSpec` and a list of `ImageConfigs` provided in the request body of the server.
- `PodConventionContextStatus` which is a status type used to represent the current status of the context retrieved by the request.

For information about an example `PodConventionContext`, see PodConventionContext in GitHub. For information about a Convention server and the structure of these types, see OpenAPI Spec in GitHub.

How the convention server works

Each convention server can host one or more conventions. The application of each convention by a convention server are controlled conditionally. The conditional criteria governing the application of a convention is customizable and are based on the evaluation of a custom Kubernetes resource called PodIntent. PodIntent is the vehicle by which Cartographer Conventions as a whole delivers its value.

A PodIntent is created, or updated if already existing, when a workload is run by using a Tanzu Application Platform supply chain. The custom resource includes both the PodTemplateSpec and the OCI image metadata associated with a workload. See the Kubernetes documentation. The conditional criteria for a convention are based on any property or value found in the PodTemplateSpec or the Open Containers Initiative (OCI) image metadata available in the PodIntent.

If a convention's criteria are met, the convention server enriches the PodTemplateSpec in the PodIntent. The convention server also updates the `status` section of the PodIntent with the name of the convention that's applied. You can figure out after the fact which conventions were applied to the workload.

To provide flexibility in how conventions are organized, you can deploy multiple convention servers. Each server can contain a convention or set of conventions focused on a specific class of runtime modifications, on a specific language framework, and so on. How the conventions are organized, grouped, and deployed is up to you and the needs of your organization.

Convention servers deployed to the cluster does not take action unless triggered to do so by the second component of Cartographer Conventions, the Convention service's controller.

Convention controller

The convention controller is the orchestrator of one or many convention servers deployed to the cluster. There are resources available on the `conventions.carto.run/v1aplha1` API that allow the controller to carry out its functions. These resources include:

- ClusterPodConvention `ClusterPodConvention` is a resource type that allows the conventions author to register a webhook server with the controller using its `spec.webhook` field.

  ```
  ...
  spec:
    selectorTarget: PodTemplateSpec # optional field with options, defaults to Po
  dTemplateSpec
    selectors: # optional, defaults to match all workloads
    - <metav1.LabelSelector>
    webhook:
      certificate:
        name: sample-cert
        namespace: sample-conventions
      clientConfig:
        <admissionregistrationv1.WebhookClientConfig>
  ```

- PodIntent

  `PodIntent` is a `conventions.carto.run/v1alpha1` resource type that is continuously reconciled and applies decorations to a workload `PodTemplateSpec` exposing the enriched `PodTemplateSpec` on its status. Whenever the status of the `PodIntent` is updated, no side effects are caused on the cluster.

As key types defined on the `conventions.carto.run` API, the ClusterPodConvention`and`PodIntent`resources are both present on the Kubernetes API Server and are queried using`clusterpodconventions.conventions.carto.run`for the former and`podintents.conventions.carto.run` for the later.

How the convention services's controller works

When the Supply Chain Choreographer creates or updates a PodIntent for a workload, the convention controller retrieves the OCI image metadata from the repository containing the workload's images and sets it in the PodIntent.

The convention controller then uses a webhook architecture to pass the PodIntent to each convention server deployed to the cluster. The controller orchestrates the processing of the PodIntent by the convention servers sequentially, based on the `priority` value that's set on the convention server. For more information, see ClusterPodConvention.

After all convention servers are finished processing a PodIntent for a workload, the convention controller updates the PodIntent with the latest version of the PodTemplateSpec and sets `PodIntent.status.conditions[].status=True` where `PodIntent.status.conditions[].type=Ready`. This status change signals the Supply Chain Choreographer that Cartographer Conventions is finished with its work. The status change also executes whatever steps are waiting in the supply chain.

Getting started

With this high-level understanding of Cartographer Conventions components, you can create and deploy a custom convention.

Note

This topic covers developing conventions using GOLANG, but this is done using other languages by following the specifications.

Prerequisites

The following prerequisites must be met before a convention is developed and deployed:

- The Kubernetes command line interface tool (kubectl) CLI is installed. For more information, see the Kubernetes documentation.
- Tanzu Application Platform prerequisites are installed. For more information, see Prerequisites
- Tanzu Application Platform components are installed. For more information, see the Installing the Tanzu CLI.
- The default supply chain is installed. Download Supply Chain Security Tools for VMware Tanzu from Tanzu Network.
- Your kubeconfig context is set to the Tanzu Application Platform-enabled cluster:

  ```
  kubectl config use-context CONTEXT_NAME
  ```

- You use GitHub to install the ko CLI. See the google/ko GitHub repository. These instructions use `ko` to build an image. If there is an existing image or build process, `ko` is optional.)

Define convention criteria

The `server.go` file contains the configuration for the server and the logic the server applies when a workload matches the defined criteria. For example, adding a Prometheus sidecar to web applications, or adding a `workload-type=spring-boot` label to any workload that has metadata, indicating it is a Spring Boot app.

Important

For this example, the package `model` defines resource types.

1. The example `server.go` configures the `ConventionHandler` to ingest the webhook requests from the convention controller. See PodConventionContext. Here the handler must only deal with the existing PodTemplateSpec and ImageConfig.

   ```
   ...
   import (
     corev1 "k8s.io/api/core/v1"
   )
   ...
   func ConventionHandler(template *corev1.PodTemplateSpec, images []model.ImageCo
   nfig) ([]string, error) {
      // Create custom conventions
   }
   ...
   ```

   Where:

   - `template` is the predefined `PodTemplateSpec` that the convention edits. For more information about `PodTemplateSpec`, see the Kubernetes documentation.

- `images` are the ImageConfig used as reference to make decisions in the conventions. In this example, the type was created within the `model` package.

2. The example `server.go` also configures the convention server to listen for requests:

```
...
import (
    "context"
    "fmt"
    "log"
    "net/http"
    "os"
    ...
)
...
func main() {
    ctx := context.Background()
    port := os.Getenv("PORT")
    if port == "" {
        port = "9000"
    }
    http.HandleFunc("/", webhook.ServerHandler(convention.ConventionHandler))
    log.Fatal(webhook.NewConventionServer(ctx, fmt.Sprintf(":%s", port)))
}
...
```

Where:

- `PORT` is a possible environment variable, for this example, defined in the Deployment.
- `ServerHandler` is the *handler* function called when any request comes to the server.
- `NewConventionServer` is the function in charge of configuring and creating the *http webhook* server.
- `port` is the calculated port of the server to listen for requests. It must match the Deployment if the `PORT` variable is not defined in it.
- The `path` or pattern (default to `/`) is the convention server's default path. If it is changed, it must be changed in the ClusterPodConvention.

Note

The *Server Handler*, `func ConventionHandler(...)`, and the configure or start web server, `func NewConventionServer(...)`, is defined in the convention controller in the `webhook` package, but you can use a custom one.

1. Creating the *Server Handler*, which handles the request from the convention controller with the PodConventionContext serialized to JSON.

```
package webhook
...
func ServerHandler(conventionHandler func(template *corev1.PodTemplateSpec, ima
ges []model.ImageConfig) ([]string, error)) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        ...
        // Check request method
        ...
        // Decode the PodConventionContext
        podConventionContext := &model.PodConventionContext{}
        err = json.Unmarshal(body, &podConventionContext)
```

```
        if err != nil {
            w.WriteHeader(http.StatusBadRequest)
            return
        }
        // Validate the PodTemplateSpec and ImageConfig
        ...
        // Apply the conventions
        pts := podConventionContext.Spec.Template.DeepCopy()
        appliedConventions, err := conventionHandler(pts, podConventionContext.
Spec.Images)
        if err != nil {
            w.WriteHeader(http.StatusInternalServerError)
            return
        }
        // Update the applied conventions and status with the new PodTemplateSp
ec
        podConventionContext.Status.AppliedConventions = appliedConventions
        podConventionContext.Status.Template = *pts
        // Return the updated PodConventionContext
        w.Header().Set("Content-Type", "application/json")
        w.WriteHeader(http.StatusOK)
        json.NewEncoder(w).Encode(podConventionContext)
    }
}
...
```

2. Configure and start the web server by defining the `NewConventionServer` function, which starts the server with the defined port and current context. The server uses the `.crt` and `.key` files to handle *TLS* traffic.

```
package webhook
...
// Watch handles the security by certificates.
type certWatcher struct {
    CrtFile string
    KeyFile string

    m       sync.Mutex
    keyPair *tls.Certificate
}
func (w *certWatcher) Load() error {
    // Creates a X509KeyPair from PEM encoded client certificate and private ke
y.
    ...
}
func (w *certWatcher) GetCertificate() *tls.Certificate {
    w.m.Lock()
    defer w.m.Unlock()

    return w.keyPair
}
...
func NewConventionServer(ctx context.Context, addr string) error {
    // Define a health check endpoint to readiness and liveness probes.
    http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) {
        w.WriteHeader(http.StatusOK)
    })

    if err := watcher.Load(); err != nil {
        return err
    }
    // Defines the server with the TLS configuration.
    server := &http.Server{
        Addr: addr,
        TLSConfig: &tls.Config{
```

```
            GetCertificate: func(_ *tls.ClientHelloInfo) (*tls.Certificate, err
or) {
                cert := watcher.GetCertificate()
                return cert, nil
            },
            PreferServerCipherSuites: true,
            MinVersion:               tls.VersionTLS13,
        },
        BaseContext: func(_ net.Listener) context.Context {
            return ctx
        },
    }
    go func() {
        <-ctx.Done()
        server.Close()
    }()

    return server.ListenAndServeTLS("", "")
}
```

Define the convention behavior

Any property or value within the PodTemplateSpec or OCI image metadata associated with a workload defines the criteria for applying conventions. See PodTemplateSpec in the Kubernetes documentation. The following are a few examples.

Matching criteria by labels or annotations

The `conventions.carto.run/v1alpha1` API allows convention authors to use the `selectorTarget` field which complements the `ClusterPodConvention` matchers to specify whether to consider labels on either one of the following available options:

- PodTemplateSpec

  ```
  ...
  template:
    metadata:
      labels:
        awesome-label: awesome-value
      annotations:
        awesome-annotation: awesome-value
  ...
  ```

- PodIntent

  ```
    ...
    kind: PodIntent
    metadata:
      name: test-pod
      labels:
        environment: production
        ...
  ```

The `selectorTarget` field is configured on the ClusterPodConvention as follows:

```
...
spec:
  selectorTarget: PodIntent # optional, defaults to PodTemplateSpec
  selectors: # optional, defaults to match all workloads
  - <metav1.LabelSelector>
  webhook:
    certificate:
```

```
    name: sample-cert
    namespace: sample-conventions
  clientConfig:
    <admissionregistrationv1.WebhookClientConfig>
```

If you do not provide a value for this optional field while using the `conventions.carto.run/v1alpha1` API, the default value is set to `PodTemplateSpec` without the conventions author explicitly doing so.

Matching criteria by environment variables

When using environment variables to define whether the convention is applicable, it must be present in the PodTemplateSpec, spec, containers, and env to validate the value.

- PodTemplateSpec

  ```
  ...
  template:
    spec:
      containers:
        - name: awesome-container
          env:
  ...
  ```

- Handler

  ```
  package convention
  ...
  func conventionHandler(template *corev1.PodTemplateSpec, images []model.ImageCo
  nfig) ([]string, error) {
      if len(template.Spec.Containers[0].Env) == 0 {
          template.Spec.Containers[0].Env = append(template.Spec.Containers[0].En
  v, corev1.EnvVar{
              Name: "MY_AWESOME_VAR",
              Value: "MY_AWESOME_VALUE",
          })
          return []string{"awesome-envs-convention"}, nil
      }
      return []string{}, nil
      ...
  }
  ```

Matching criteria by image metadata

For each image contained within the PodTemplateSpec, the convention controller fetches the OCI image metadata and known bill of materials (BOMs), providing it to the convention server as ImageConfig. This metadata is introspected to make decisions about how to configure the PodTemplateSpec.

Configure and install the convention server

The `server.yaml` defines the Kubernetes components that enable the convention server in the cluster. The next definitions are within the file.

1. A `namespace` is created for the convention server components and has the required objects to run the server. It's used in the ClusterPodConvention section to indicate to the controller where the server is.

   ```
   ...
   ---
   apiVersion: v1
   kind: Namespace
   ```

```
metadata:
  name: awesome-convention
---
...
```

2. (Optional) A certificate manager `Issuer` is created to issue the> certificate needed for TLS communication.

```
...
---
# The following manifests contain a self-signed issuer CR and a certificate CR.
# More document can be found at https://docs.cert-manager.io
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: awesome-selfsigned-issuer
  namespace: awesome-convention
spec:
  selfSigned: {}
---
...
```

3. (Optional) A self-signed `Certificate` is created.

```
...
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: awesome-webhook-cert
  namespace: awesome-convention
spec:
  subject:
    organizations:
    - vmware
    organizationalUnits:
    - tanzu
  commonName: awesome-webhook.awesome-convention.svc
  dnsNames:
  - awesome-webhook.awesome-convention.svc
  - awesome-webhook.awesome-convention.svc.cluster.local
  issuerRef:
    kind: Issuer
    name: awesome-selfsigned-issuer
  secretName: awesome-webhook-cert
  revisionHistoryLimit: 10
---
...
```

4. A Kubernetes `Deployment` is created to run the webhook from. The Service uses the container port defined by the `Deployment` to expose the server.

```
...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: awesome-webhook
  namespace: awesome-convention
spec:
  replicas: 1
  selector:
    matchLabels:
    app: awesome-webhook
```

```
  template:
    metadata:
      labels:
        app: awesome-webhook
    spec:
      containers:
      - name: webhook
        # Set the prebuilt image of the convention or use ko to build an image
from code.
        # see https://github.com/google/ko
        image: ko://awesome-repo/awesome-user/awesome-convention
      env:
      - name: PORT
        value: "8443"
      ports:
      - containerPort: 8443
        name: webhook
      livenessProbe:
        httpGet:
          scheme: HTTPS
          port: webhook
          path: /healthz
      readinessProbe:
        httpGet:
          scheme: HTTPS
          port: webhook
          path: /healthz
      volumeMounts:
      - name: certs
        mountPath: /config/certs
        readOnly: true
    volumes:
    - name: certs
      secret:
        defaultMode: 420
        secretName: awesome-webhook-cert
---
...
```

5. A Kubernetes `Service` to expose the convention deployment is created. For this example, the exposed port is the default `443`. If you change the port, the ClusterPodConvention must be updated.

```
...
---
apiVersion: v1
kind: Service
metadata:
  name: awesome-webhook
  namespace: awesome-convention
  labels:
    app: awesome-webhook
spec:
  selector:
    app: awesome-webhook
  ports:
    - protocol: TCP
      port: 443
      targetPort: webhook
---
...
```

6. The ClusterPodConvention adds the convention to the cluster to make it available for the convention controller:

Important

The `annotations` block is only needed if you use a self-signed certificate. See the cert-manager documentation.

```
...
---
apiVersion: conventions.carto.run/v1alpha1
kind: ClusterPodConvention
metadata:
  name: awesome-convention
  annotations:
    conventions.carto.run/inject-ca-from: "awesome-convention/awesome-webhook-c
ert"
spec:
  webhook:
    clientConfig:
      service:
        name: awesome-webhook
        namespace: awesome-convention
        # path: "/" # default
        # port: 443 # default
```

Deploy a convention server

To deploy a convention server:

1. Build and install the convention.
    - To build and deploy the convention, use the ko tool on GitHub. It compiles your Go code into a Docker image and pushes it to the registry `KO_DOCKER_REGISTRY`.

      ```
      ko apply -f dist/server.yaml
      ```

    - If a different tool builds the image, the configuration is also applied by using either kubectl or `kapp`, setting the correct image in the Deployment descriptor.

      kubectl

      ```
      kubectl apply -f server.yaml
      ```

      kapp

      ```
      kapp deploy -y -a awesome-convention -f server.yaml
      ```

2. Verify the convention server. To verify the status of the convention server, confirm the running convention pods:
    - If the server is running, `kubectl get all -n awesome-convention` returns output such as:

      ```
      NAME                                    READY   STATUS    RESTARTS   A
      GE
      pod/awesome-webhook-1234567890-12345    1/1     Running   0          8
      h

      NAME                          TYPE        CLUSTER-IP    EXTERNAL-IP   POR
      T(S)   AGE
      service/awesome-webhook       ClusterIP   10.56.12.49   <none>        44
      3/TCP   28h
      ```

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AG
E
deployment.apps/awesome-webhook         1/1     1            1           28
h

NAME                                              DESIRED   CURRENT   READ
Y   AGE
replicaset.apps/awesome-webhook-1234563213        0         0         0
23h
replicaset.apps/awesome-webhook-5b79d5cb59        0         0         0
28h
replicaset.apps/awesome-webhook-5bf557c9f8        1         1         1
20h
replicaset.apps/awesome-webhook-77c647c987        0         0         0
23h
replicaset.apps/awesome-webhook-79d9c6f74c        0         0         0
23h
replicaset.apps/awesome-webhook-7d9d667b8d        0         0         0
9h
replicaset.apps/awesome-webhook-8668664d75        0         0         0
23h
replicaset.apps/awesome-webhook-9b6957476         0         0         0
24h
```

- To verify that the conventions are applied, ensure that the `PodIntent` of a workload that matches the convention criteria:

```
kubectl -o yaml get podintents.conventions.apps.tanzu.vmware.co awesome-a
pp
```

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2021-10-07T13:30:00Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    carto.run/cluster-supply-chain-name: awesome-supply-chain
    carto.run/cluster-template-name: convention-template
    carto.run/component-name: config-provider
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: awesome-app
    carto.run/workload-namespace: default
  name: awesome-app
  namespace: default
ownerReferences:
- apiVersion: carto.run/v1alpha1
  blockOwnerDeletion: true
  controller: true
  kind: Workload
  name: awesome-app
  uid: "********"
resourceVersion: "********"
uid: "********"
spec:
imagePullSecrets:
  - name: registry-credentials
    serviceAccountName: default
    template:
      metadata:
        annotations:
          developer.conventions/target-containers: workload
        labels:
          app.kubernetes.io/component: run
```

```
          app.kubernetes.io/part-of: awesome-app
          carto.run/workload-name: awesome-app
      spec:
        containers:
        - image: awesome-repo.com/awesome-project/awesome-app@sha256:****
****
          name: workload
          resources: {}
          securityContext:
          runAsUser: 1000
status:
  conditions:
  - lastTransitionTime: "2021-10-07T13:30:00Z"
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2021-10-07T13:30:00Z"
    status: "True"
    type: Ready
observedGeneration: 1
template:
  metadata:
    annotations:
      awesome-annotation: awesome-value
      conventions.carto.run/applied-conventions: |-
        awesome-label-convention
        awesome-annotation-convention
        awesome-envs-convention
        awesome-image-convention
        developer.conventions/target-containers: workload
    labels:
      awesome-label: awesome-value
      app.kubernetes.io/component: run
      app.kubernetes.io/part-of: awesome-app
      carto.run/workload-name: awesome-app
      conventions.carto.run/framework: go
  spec:
    containers:
    - env:
      - name: MY_AWESOME_VAR
        value: "MY_AWESOME_VALUE"
      image: awesome-repo.com/awesome-project/awesome-app@sha256:********
      name: workload
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      securityContext:
        runAsUser: 1000
```

Next Steps

Keep Exploring:

- Try to use different matching criteria for the conventions or enhance the supply chain with multiple conventions.

Troubleshoot Cartographer Conventions

This topic describes how you can troubleshoot Cartographer Conventions.

No server in the cluster

Symptoms

- When a `PodIntent` is submitted, no `convention` is applied.

Cause

When there are no `convention servers` (ClusterPodConvention) deployed in the cluster or none of the existing convention servers applied any conventions, the `PodIntent` is not being mutated.

Solution

Deploy a `convention server` (ClusterPodConvention) in the cluster.

Server with wrong certificates configured

Symptoms

- When a `PodIntent` is submitted, the `conventions` are not applied.
- The `convention-controller` logs reports an error `failed to get CABundle` as follows:


```
{
"level": "error",
"ts": 1638222343.6839523,
"logger": "controllers.PodIntent.PodIntent.ResolveConventions",
"msg": "failed to get CABundle",
"ClusterPodConvention": "base-convention",
"error": "unable to find valid certificaterequests for certificate \"convention
-template/webhook-certificate\"",
"stacktrace": "reflect.Value.Call\n\treflect/value.go:339\ngithub.com/vmware-la
bs/reconciler-runtime/reconcilers.(*SyncReconciler).sync\n\tgithub.com/vmware-l
abs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:287\ngithub.com/vmware
-labs/reconciler-runtime/reconcilers.(*SyncReconciler).Reconcile\n\tgithub.com/
vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:276\ngithub.co
m/vmware-labs/reconciler-runtime/reconcilers.Sequence.Reconcile\n\tgithub.com/v
mware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:815\ngithub.co
m/vmware-labs/reconciler-runtime/reconcilers.(*ParentReconciler).reconcile\n\tg
ithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:146
\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*ParentReconciler).Rec
oncile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconcil
ers.go:120\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controlle
r).Reconcile\n\tsigs.k8s.io/controller-runtime@v0.10.3/pkg/internal/controller/
controller.go:114\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Con
troller).reconcileHandler\n\tsigs.k8s.io/controller-runtime@v0.10.3/pkg/interna
l/controller/controller.go:311\nsigs.k8s.io/controller-runtime/pkg/internal/con
troller.(*Controller).processNextWorkItem\n\tsigs.k8s.io/controller-runtime@v0.
10.3/pkg/internal/controller/controller.go:266\nsigs.k8s.io/controller-runtime/
pkg/internal/controller.(*Controller).Start.func2.2\n\tsigs.k8s.io/controller-r
untime@v0.10.3/pkg/internal/controller/controller.go:227"
```


Cause

`convention server` (ClusterPodConvention) is configured with wrong certificates. The `convention-controller` cannot figure out the CA Bundle to perform the request to the server.

Solution

Ensure that the `convention server` (ClusterPodConvention) is configured with the correct certificates. To do so, verify the value of annotation `conventions.carto.run/inject-ca-from` which must be set to the used Certificate.

Important

Do not set annotation `conventions.carto.run/inject-ca-from` if no certificate is used.

Server fails when processing a request

Symptoms

- When a `PodIntent` is submitted, the `convention` is not applied.
- The `convention-controller` logs reports `failed to apply convention` error like this.

```
{"level":"error","ts":1638205387.8813763,"logger":"controllers.PodIntent.PodInt
ent.ApplyConventions","msg":"failed to apply convention","Convention":{"Nam
e":"base-convention","Selectors":null,"Priority":"Normal","ClientConfig":{"serv
ice":{"namespace":"convention-template","name":"webhook","port":443},"caBundl
e":"..."}},"error":"Post \"https://webhook.convention-template.svc:443/?timeout
=30s\": EOF","stacktrace":"reflect.Value.call\n\treflect/value.go:543\nreflect.
Value.Call\n\treflect/value.go:339\ngithub.com/vmware-labs/reconciler-runtime/r
econcilers.(*SyncReconciler).sync\n\tgithub.com/vmware-labs/reconciler-runtime@
v0.3.0/reconcilers/reconcilers.go:287\ngithub.com/vmware-labs/reconciler-runtim
e/reconcilers.(*SyncReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-
runtime@v0.3.0/reconcilers/reconcilers.go:276\ngithub.com/vmware-labs/reconcile
r-runtime/reconcilers.Sequence.Reconcile\n\tgithub.com/vmware-labs/reconciler-r
untime@v0.3.0/reconcilers/reconcilers.go:815\ngithub.com/vmware-labs/reconciler
-runtime/reconcilers.(*ParentReconciler).reconcile\n\tgithub.com/vmware-labs/re
conciler-runtime@v0.3.0/reconcilers/reconcilers.go:146\ngithub.com/vmware-labs/
reconciler-runtime/reconcilers.(*ParentReconciler).Reconcile\n\tgithub.com/vmwa
re-labs/reconciler-runtime@v0.3.0/reconcilers/reconcilers.go:120\nsigs.k8s.io/c
ontroller-runtime/pkg/internal/controller.(*Controller).Reconcile\n\tsigs.k8s.i
o/controller-runtime@v0.10.0/pkg/internal/controller/controller.go:114\nsigs.k8
s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler
\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/internal/controller/controller.go
:311\nsigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).pro
cessNextWorkItem\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/internal/control
ler/controller.go:266\nsigs.k8s.io/controller-runtime/pkg/internal/controller.
(*Controller).Start.func2.2\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/inter
nal/controller/controller.go:227"}
```

- When a `PodIntent` status message is updated with `failed to apply convention from source base-convention: Post "https://webhook.convention-template.svc:443/?timeout=30s": EOF`.

Cause

An unmanaged error occurs in the `convention server` when processing a request.

Solution

1. Check the `convention server` logs to identify the cause of the error:
    1. Use the following command to retrieve the `convention server` logs:

        ```
        kubectl -n convention-template logs deployment/webhook
        ```

        Where:

        - The convention server was deployed as a `Deployment`
        - `webhook` is the name of the convention server `Deployment`.

    - `convention-template` is the namespace where the convention server is deployed.

2. Identify the error and deploy a fixed version of `convention server`.
    - Be aware that the new deployment is not applied to the existing `PodIntent`s. It is only applied to the new `PodIntent`s.
    - To apply new deployment to exiting `PodIntent`, you must update the `PodIntent`, so the reconciler applies if it matches the criteria.

Connection refused due to unsecured connection

Symptoms

- When a `PodIntent` is submitted, the `convention` is not applied.
- The `convention-controller` logs reports a connection refused error as follows:

```
{"level":"error","ts":1638202791.5734537,"logger":"controllers.PodIntent.PodInt
ent.ApplyConventions","msg":"failed to apply convention","Convention":{"Nam
e":"base-convention","Selectors":null,"Priority":"Normal","ClientConfig":{"serv
ice":{"namespace":"convention-template","name":"webhook","port":443},"caBundl
e":"..."}},"error":"Post \"https://webhook.convention-template.svc:443/?timeout
=30s\": dial tcp 10.56.13.206:443: connect: connection refused","stacktrace":"r
eflect.Value.call\n\treflect/value.go:543\nreflect.Value.Call\n\treflect/value.
go:339\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*SyncReconcile
r).sync\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconcilers/reconci
lers.go:287\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*SyncReconc
iler).Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconciler
s/reconcilers.go:276\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.Seq
uence.Reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/reconciler
s/reconcilers.go:815\ngithub.com/vmware-labs/reconciler-runtime/reconcilers.(*P
arentReconciler).reconcile\n\tgithub.com/vmware-labs/reconciler-runtime@v0.3.0/
reconcilers/reconcilers.go:146\ngithub.com/vmware-labs/reconciler-runtime/recon
cilers.(*ParentReconciler).Reconcile\n\tgithub.com/vmware-labs/reconciler-runti
me@v0.3.0/reconcilers/reconcilers.go:120\nsigs.k8s.io/controller-runtime/pkg/in
ternal/controller.(*Controller).Reconcile\n\tsigs.k8s.io/controller-runtime@v0.
10.0/pkg/internal/controller/controller.go:114\nsigs.k8s.io/controller-runtime/
pkg/internal/controller.(*Controller).reconcileHandler\n\tsigs.k8s.io/controlle
r-runtime@v0.10.0/pkg/internal/controller/controller.go:311\nsigs.k8s.io/contro
ller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem\n\tsigs.
k8s.io/controller-runtime@v0.10.0/pkg/internal/controller/controller.go:266\nsi
gs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2.
2\n\tsigs.k8s.io/controller-runtime@v0.10.0/pkg/internal/controller/controller.
go:227"}
```

- The `convention server` fails to start due to `server gave HTTP response to HTTPS client`:
    - When checking the `convention server` events by running the following command:

```
kubectl -n convention-template describe pod webhook-594d75d69b-4w4s8
```

    Where:

    - The convention server was deployed as a `Deployment`
    - `webhook-594d75d69b-4w4s8` is the name of the `convention server` Pod.
    - `convention-template` is the namespace where the convention server is deployed.

    For example:

```
Name:         webhook-594d75d69b-4w4s8
Namespace:    convention-template
```

```
...
Containers:
  webhook:
...
Events:
Type     Reason     Age                   From               Message
----     ------     ----                  ----               -------
Normal   Scheduled  14m                   default-scheduler  Successfully assig
ned convention-template/webhook-594d75d69b-4w4s8 to pool
Normal   Pulling    14m                   kubelet            Pulling image "awe
some-repo/awesome-user/awesome-convention-..."
Normal   Pulled     14m                   kubelet            Successfully pulle
d image "awesome-repo/awesome-user/awesome-convention..." in 1.06032653s
Normal   Created    13m (x2 over 14m)     kubelet            Created container
webhook
Normal   Started    13m (x2 over 14m)     kubelet            Started container
webhook
Warning  Unhealthy  13m (x9 over 14m)     kubelet            Readiness probe fa
iled: Get "https://10.52.2.74:8443/healthz": http: server gave HTTP response to
HTTPS client
Warning  Unhealthy  13m (x6 over 14m)     kubelet            Liveness probe fai
led: Get "https://10.52.2.74:8443/healthz": http: server gave HTTP response to
HTTPS client
Normal   Pulled     9m13s (x6 over 13m)   kubelet            Container image "a
wesome-repo/awesome-user/awesome-convention" already present on machine
Warning  BackOff    4m22s (x32 over 11m)  kubelet            Back-off restartin
g failed container
```

Cause

When a `convention server` is provided without using Transport Layer Security (TLS) but the `Deployment` is configured to use TLS, Kubernetes fails to deploy the `Pod` because of the `liveness probe`.

Solution

1. Deploy a `convention server` with TLS enabled.
2. Create `ClusterPodConvention` resource for the convention server with annotation `conventions.carto.run/inject-ca-from` as a pointer to the deployed `Certificate` resource.

Self-signed certificate authority (CA) not propagated to the Convention Service

Symptoms

The self-signed certificate authority (CA) for a registry is not propagated to the Convention Service.

Cause

When you provide the self-signed certificate authority (CA) for a registry through `convention-controller.ca_cert_data`, it cannot be propagated to the Convention Service.

Solution

Define the CA by using the available `.shared.ca_cert_data` top-level key to supply the CA to the Convention Service.

No imagePullSecrets configured

Symptoms

When a PodIntent is submitted:

- No convention is applied.
- You see an `unauthorized to access repository` or `fetching metadata for Images failed` error when you inspect the workload.

Cause

The errors are seen when a `workload` is created in a developer namespace where `imagePullSecrets` are not defined on the `default` serviceAccount or on the preferred serviceAccount.

Solution

Add the `imagePullSecrets` name to the default serviceAccount or the preferred serviceAccount.

For example:

```
kind: ServiceAccount
metadata:
  name: default
  namespace: my-workload-namespace
imagePullSecrets:
  - name: registry-credentials # ensure this secret is defined
secrets:
- name: registry-credentials
```

Convention Service Resources for Cartographer Conventions

This reference topic describes the convention service resources you can use with Cartographer Conventions.

Overview

There are several resources involved in the application of conventions to workloads and these are typically consumed by platform developers and operators rather than by application developers.

- ClusterPodConvention

  The following is an example `conventions.carto.run/v1alpha1` type:

  ```
  ---
  apiVersion: conventions.carto.run/v1alpha1
  kind: ClusterPodConvention
  metadata:
  name: sample
  spec:
  selectorTarget: PodTemplateSpec # optional field with options, defaults to PodT
  emplateSpec
  selectors: # optional, defaults to match all workloads
  - <metav1.LabelSelector>
  webhook:
    certificate:
      name: sample-cert
      namespace: sample-conventions
    clientConfig:
      <admissionregistrationv1.WebhookClientConfig>
  ```

A `ClusterPodConvention` can target a one or more workloads of different types. You can apply multiple conventions to a single workload. It is at the discretion of the “Conventions Author” how a convention is applied.

To list out available conventions in your cluster, run the following `kubectl`command

```
$ kubectl get clusterpodconventions.conventions.carto.run

  NAME                      AGE
  appliveview-sample        23h
  developer-conventions     23h
  spring-boot-convention    23h
```

- PodIntent

  The following is an example `conventions.carto.run/v1alpha1` resource:

  ```
  apiVersion: conventions.carto.run/v1alpha1
  kind: PodIntent
  metadata:
  name: sample
  spec:
  imagePullSecrets: <[]corev1.LocalObjectReference> # optional
  serviceAccountName: <string> # optional, defaults to 'default'
  template:
    <corev1.PodTemplateSpec>
  status:
  observedGeneration: 1 # reflected from .metadata.generation
  conditions:
  - <metav1.Condition>
  template: # enriched PodTemplateSpec
    <corev1.PodTemplateSpec>
  ```

  To list out available `PodIntent` resources in your cluster, run the following kubectl command

  ```
  # specify relevant namespace
  kubectl get podintents.conventions.carto.run -n my-apps

  NAME             READY   REASON               AGE
  spring-sample    True    ConventionsApplied   8m5s
  ```

  When a`PodIntent` is created, the `PodIntent` reconciler lists all `ClusterPodConventions` resources and applies them serially. To ensure that the consistency of the enriched PodTemplateSpec, the list of `ClusterPodConventions`is sorted alphabetically by name before applying the conventions.

  Tip : *You can use strategic naming to control the order in which the conventions are applied.*

  After the conventions are applied, the `Ready` status condition on the `PodIntent` resource is used to indicate whether it is applied. A list of all applied conventions is stored under the annotation `conventions.carto.run/applied-conventions`.

There are also a few other resources available to the `Conventions Author` that are not persisted in your cluster, including:

- ImageConfig
- PodConventionContextSpec
- PodConventionContextStatus
- PodConventionContext

- BOM

Collecting Logs from the Controller

A successful deployment of the convention service creates it's resources on the following `cartographer-system` namespace:

```
$ kubectl get all -n cartographer-system
  NAME                                                                READY   STATUS
RESTARTS   AGE
  ...
  pod/cartographer-conventions-controller-manager-76fd86789f-lzh86   1/1     Running
0          20h

  NAME                                                                   TYPE        CL
USTER-IP     EXTERNAL-IP   PORT(S)    AGE
  ...
  service/cartographer-conventions-controller-manager-metrics-service   ClusterIP   1
0.0.250.231   <none>        8443/TCP   20h
  service/cartographer-conventions-webhook-service                      ClusterIP   1
0.0.6.254     <none>        443/TCP    20h
  ...

  NAME                                                          READY   UP-TO-DATE   A
VAILABLE   AGE
  ...
  deployment.apps/cartographer-conventions-controller-manager   1/1     1            1
20h

  NAME                                                                      DESIRED   C
URRENT   READY   AGE
  ...
  replicaset.apps/cartographer-conventions-controller-manager-76fd86789f   1         1
1       20h
```

In order to examine logs from the cartographer conventions controller to help identify issues, inspect the cartographer conventions controller manager pod as follows

```
 kubectl -n cartographer-system logs -l control-plane=controller-manager
...
{"level":"info","ts":"2023-02-06T20:49:19.855086032Z","logger":"MetricsReconciler","ms
g":"reconciling builders configmap","controller":"configmap","controllerGroup":"","con
trollerKind":"ConfigMap","ConfigMap":{"name":"controller-manager-metrics-data","namesp
ace":"cartographer-system"},"namespace":"cartographer-system","name":"controller-manag
er-metrics-data","reconcileID":"6f5e38c7-0ce0-4c74-aff3-f938fb742dab","diff":"  map[st
ring]string{\n- \t\"clusterpodconventions_names\": \"appliveview-sample\",\n+ \t\"clus
terpodconventions_names\": \"appliveview-sample\nspring-boot-convention\",\n  \t\"podi
ntents_count\":            \"0\",\n  }\n"}
{"level":"info","ts":"2023-02-06T20:49:20.101742252Z","logger":"MetricsReconciler","ms
g":"reconciling builders configmap","controller":"configmap","controllerGroup":"","con
trollerKind":"ConfigMap","ConfigMap":{"name":"controller-manager-metrics-data","namesp
ace":"cartographer-system"},"namespace":"cartographer-system","name":"controller-manag
er-metrics-data","reconcileID":"3a1950bc-4c55-47bb-8380-2de574bd5d5e","diff":"  map[st
ring]string{\n  \t\"clusterpodconventions_names\": strings.Join({\n  \t\t\"appliveview
-sample\n\",\n+ \t\t\"developer-conventions\n\",\n  \t\t\"spring-boot-convention\",\n
\t}, \"\"),\n  \t\"podintents_count\": \"0\",\n  }\n"}
...
```

Convention Service Resources for Cartographer Conventions

This reference topic describes the convention service resources you can use with Cartographer Conventions.

Overview

There are several resources involved in the application of conventions to workloads and these are typically consumed by platform developers and operators rather than by application developers.

- ClusterPodConvention

  The following is an example `conventions.carto.run/v1alpha1` type:

  ```
  ---
  apiVersion: conventions.carto.run/v1alpha1
  kind: ClusterPodConvention
  metadata:
  name: sample
  spec:
  selectorTarget: PodTemplateSpec # optional field with options, defaults to PodT
  emplateSpec
  selectors: # optional, defaults to match all workloads
  - <metav1.LabelSelector>
  webhook:
    certificate:
      name: sample-cert
      namespace: sample-conventions
    clientConfig:
      <admissionregistrationv1.WebhookClientConfig>
  ```

  A `ClusterPodConvention` can target a one or more workloads of different types. You can apply multiple conventions to a single workload. It is at the discretion of the “Conventions Author” how a convention is applied.

  To list out available conventions in your cluster, run the following `kubectl`command

  ```
  $ kubectl get clusterpodconventions.conventions.carto.run

    NAME                      AGE
    appliveview-sample        23h
    developer-conventions     23h
    spring-boot-convention    23h
  ```

- PodIntent

  The following is an example `conventions.carto.run/v1alpha1` resource:

  ```
  apiVersion: conventions.carto.run/v1alpha1
  kind: PodIntent
  metadata:
  name: sample
  spec:
  imagePullSecrets: <[]corev1.LocalObjectReference> # optional
  serviceAccountName: <string> # optional, defaults to 'default'
  template:
    <corev1.PodTemplateSpec>
  status:
  observedGeneration: 1 # reflected from .metadata.generation
  conditions:
  - <metav1.Condition>
  template: # enriched PodTemplateSpec
    <corev1.PodTemplateSpec>
  ```

To list out available `PodIntent` resources in your cluster, run the following kubectl command

```
# specify relevant namespace
kubectl get podintents.conventions.carto.run -n my-apps

NAME            READY   REASON               AGE
spring-sample   True    ConventionsApplied   8m5s
```

When a`PodIntent` is created, the `PodIntent` reconciler lists all `ClusterPodConventions` resources and applies them serially. To ensure that the consistency of the enriched PodTemplateSpec, the list of `ClusterPodConventions`is sorted alphabetically by name before applying the conventions.

Tip : *You can use strategic naming to control the order in which the conventions are applied.*

After the conventions are applied, the `Ready` status condition on the `PodIntent` resource is used to indicate whether it is applied. A list of all applied conventions is stored under the annotation `conventions.carto.run/applied-conventions`.

There are also a few other resources available to the `Conventions Author` that are not persisted in your cluster, including:

- ImageConfig
- PodConventionContextSpec
- PodConventionContextStatus
- PodConventionContext
- BOM

Collecting Logs from the Controller

A successful deployment of the convention service creates it's resources on the following `cartographer-system` namespace:

```
$ kubectl get all -n cartographer-system
  NAME                                                              READY   STATUS
RESTARTS   AGE
  ...
  pod/cartographer-conventions-controller-manager-76fd86789f-lzh86   1/1     Running
0          20h

  NAME                                                                 TYPE        CL
USTER-IP     EXTERNAL-IP   PORT(S)    AGE
  ...
  service/cartographer-conventions-controller-manager-metrics-service   ClusterIP   1
0.0.250.231   <none>        8443/TCP   20h
  service/cartographer-conventions-webhook-service                      ClusterIP   1
0.0.6.254     <none>        443/TCP    20h
  ...

  NAME                                                         READY   UP-TO-DATE   A
VAILABLE   AGE
  ...
  deployment.apps/cartographer-conventions-controller-manager   1/1     1            1
20h

  NAME                                                                    DESIRED   C
URRENT   READY   AGE
  ...
```

```
  replicaset.apps/cartographer-conventions-controller-manager-76fd86789f   1         1
1       20h
```

In order to examine logs from the cartographer conventions controller to help identify issues, inspect the cartographer conventions controller manager pod as follows

```
 kubectl -n cartographer-system logs -l control-plane=controller-manager
...
{"level":"info","ts":"2023-02-06T20:49:19.855086032Z","logger":"MetricsReconciler","ms
g":"reconciling builders configmap","controller":"configmap","controllerGroup":"","con
trollerKind":"ConfigMap","ConfigMap":{"name":"controller-manager-metrics-data","namesp
ace":"cartographer-system"},"namespace":"cartographer-system","name":"controller-manag
er-metrics-data","reconcileID":"6f5e38c7-0ce0-4c74-aff3-f938fb742dab","diff":"  map[st
ring]string{\n- \t\"clusterpodconventions_names\": \"appliveview-sample\",\n+ \t\"clus
terpodconventions_names\": \"appliveview-sample\nspring-boot-convention\",\n  \t\"podi
ntents_count\":            \"0\",\n  }\n"}
{"level":"info","ts":"2023-02-06T20:49:20.101742252Z","logger":"MetricsReconciler","ms
g":"reconciling builders configmap","controller":"configmap","controllerGroup":"","con
trollerKind":"ConfigMap","ConfigMap":{"name":"controller-manager-metrics-data","namesp
ace":"cartographer-system"},"namespace":"cartographer-system","name":"controller-manag
er-metrics-data","reconcileID":"3a1950bc-4c55-47bb-8380-2de574bd5d5e","diff":"  map[st
ring]string{\n  \t\"clusterpodconventions_names\": strings.Join({\n  \t\t\"appliveview
-sample\n\",\n+ \t\t\"developer-conventions\n\",\n  \t\t\"spring-boot-convention\",\n
\t}, \"\"),\n  \t\"podintents_count\": \"0\",\n  }\n"}
...
```

ImageConfig for Cartographer Conventions

This reference topic describes the ImageConfig object you can use with Cartographer Conventions.

Overview

The image configuration object holds the name of the image, the BOM, and the OCI image configuration with image metadata from the repository.

OCI image configuration contains the metadata from the image repository.

The BOM represents the content of the image and may be zero or more per image.

```
{
  "name": "oci-image-name",
  "boms": [{
      "name": "bom-name",
      "raw": "`a byte array`"
  }],
  "config": {
      {
        "created": "2015-10-31T22:22:56.015925234Z",
        "author": "Alyssa P. Hacker <alyspdev@example.com>",
        "architecture": "amd64",
        "os": "linux",
        "config": {
            "User": "alice",
            "ExposedPorts": {
                "8080/tcp": {}
            },
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "FOO=oci_is_a",
                "BAR=well_written_spec"
            ],
            "Entrypoint": [
                "/bin/my-app-binary"
```

```
            ],
            "Cmd": [
                "--foreground",
                "--config",
                "/etc/my-app.d/default.cfg"
            ],
            "Volumes": {
                "/var/job-result-data": {},
                "/var/log/my-app-logs": {}
            },
            "WorkingDir": "/home/alice",
            "Labels": {
                "com.example.project.git.url": "https://example.com/project.git",
                "com.example.project.git.commit": "45a939b2999782a3f005621a8d0f29aa387
e1d6b"
            }
        },
        "rootfs": {
        "diff_ids": [
            "sha256:c6f988f4874bb0add23a778f753c65efe992244e148a1d2ec2a8b664fb66bbd1",
            "sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef"
        ],
        "type": "layers"
        },
        "history": [
        {
            "created": "2015-10-31T22:22:54.690851953Z",
            "created_by": "/bin/sh -c #(nop) ADD file:a3bc1e842b69636f9df5256c49c5374f
b4eef1e281fe3f282c65fb853ee171c5 in /"
        },
        {
            "created": "2015-10-31T22:22:55.613815829Z",
            "created_by": "/bin/sh -c #(nop) CMD [\"sh\"]",
            "empty_layer": true
        },
        {
            "created": "2015-10-31T22:22:56.329850019Z",
            "created_by": "/bin/sh -c apk add curl"
        }
        ]
    }
  }
}
```

PodConventionContextSpec for Cartographer Conventions

This reference topic describes the `PodConventionContextSpec` you can use with Cartographer Conventions.

Overview

The Pod convention context specification is a wrapper of the PodTemplateSpec and the ImageConfig provided in the request body of the server. It represents the original `PodTemplateSpec`. For more information on `PodTemplateSpec`, see the Kubernetes documentation.

```
{
"template": {
    "metadata": {
        ...
    },
    "spec": {
        ...
    }
```

```
},
"imageConfig": {
    ...
  "name": "oci-image-name",
  "config": {
        ...
    }
  }
}
```

PodConventionContextStatus for Cartographer Conventions

This reference topic describes the `PodConventionContextStatus` status type that you can use with Cartographer Conventions.

Overview

The Pod convention context status type is used to represent the current status of the context retrieved by the request. It holds the applied conventions by the server and the modified version of the `PodTemplateSpec`. For more information about `PodTemplateSpec`, see the Kubernetes documentation.

The field `.template` is populated with the enriched PodTemplateSpec. The field `.appliedConventions` is populated with the names of any applied conventions.

```
{
    "template": {
        "metadata": {
            ...
        },
        "spec": {
            ...
        }
    },
    "appliedConventions": [
        "convention-1",
        "convention-2",
        "convention-4"
    ]
}
```

yaml version:

```
---
apiVersion: webhooks.conventions.carto.run/v1alpha1
kind: PodConventionContext
metadata:
  name: sample # the name of the ClusterPodConvention
spec: # the request
  imageConfig:
  template:
    <corev1.PodTemplateSpec>
status: # the response
  appliedConventions: # list of names of conventions applied
  - my-convention
  template:
  spec:
      containers:
      - name : workload
        image: helloworld-go-mod
```

PodConventionContext for Cartographer Conventions

This reference topic describes the `PodConventionContext` that you can use with Cartographer Conventions.

Overview

The Pod convention context is the body of the webhook request and response. The specification is provided by the convention controller and the status is set by the convention server.

The context is a wrapper of the individual object description in an API (TypeMeta), the persistent metadata of a resource (ObjectMeta), the PodConventionContextSpec and the PodConventionContextStatus.

PodConventionContext Objects

In the `PodConventionContext` API resource:

- Object path `.spec.template` field defines the `PodTemplateSpec` to be enriched by conventions. For more information about `PodTemplateSpec`, see the Kubernetes documentation.
- Object path `.spec.imageConfig[]` field defines ImageConfig. Each entry of it is populated with the name of the image (`.spec.imageConfig[].image`) and its OCI metadata (`.spec.imageConfig[].config`). These entries are generated for each image referenced in PodTemplateSpec (`.spec.template`).

The following is an example of a `PodConventionContext` resource request received by the convention server. This resource is generated for a Go language-based application image in GitHub. It is built with Cloud Native Paketo Buildpacks that use Go mod for dependency management.

```
---
apiVersion: webhooks.conventions.carto.run/v1alpha1
kind: PodConventionContext
metadata:
  name: sample # the name of the ClusterPodConvention
spec: # the request
  imageConfig: # one entry per image referenced by the PodTemplateSpec
  - image: sample/go-based-image
    boms:
    - name: cnb-app:.../sbom.cdx.json
      raw: ...
    config:
      entrypoint:
      - "/cnb/process/web"
      domainname: ""
      architecture: "amd64"
      image: "sha256:05b698a4949db54fdb36ea431477867abf51054abd0cbfcfd1bb81cda1842288"
      labels:
        "io.buildpacks.stack.distro.version": "18.04"
        "io.buildpacks.stack.homepage": "https://github.com/paketo-buildpacks/stacks"
        "io.buildpacks.stack.id": "io.buildpacks.stacks.bionic"
        "io.buildpacks.stack.maintainer": "Paketo Buildpacks"
        "io.buildpacks.stack.distro.name": "Ubuntu"
        "io.buildpacks.stack.metadata": `{"app":[{"sha":"sha256:ea4ec23266a3af1204fd64
3de0f3572dd8dbb5697a5ef15bdae844777c19bf8f"}],
        "buildpacks":[{"key":"paketo-buildpac`...,
        "io.buildpacks.build.metadata": `{"bom":[{"name":"go","metadata":{"licenses":
[],"name":"Go","sha256":"7fef8ba6a0786143efcce66b0bbfbfbab02afeef522b4e09833c5b550d7
`...
  template:
```

```
    spec:
      containers:
      - name : workload
        image: helloworld-go-mod
```

PodConventionContext Structure

This section introduces more information about the image configuration in `PodConventionContext`. The convention-controller passes this information for each image in good faith. The controller is not the source of the metadata, and there is no guarantee that the information is correct.

The `config` field in the image configuration passes through the OCI Image metadata in GitHub loaded from the registry for the image.

The `boms` field in the image configuration passes through the BOMs of the image. Conventions might parse the BOMs they want to inspect. There is no guarantee that an image contains a BOM or that the BOM is in a certain format.

ClusterPodConvention for Cartographer Conventions

This reference topic describes the `ClusterPodConvention` that you can use with Cartographer Conventions.

Overview

A `ClusterPodConvention` defines how to connect to convention servers. It provides a way to apply a set of conventions to a PodTemplateSpec and artifact metadata. A convention typically focuses on a particular application framework, but might be cross cutting. Applied conventions must be pure functions.

Define conventions

Webhook servers are the only way to define conventions.

```
apiVersion: conventions.carto.run/v1alpha1
kind: ClusterPodConvention
metadata:
  name: base-convention
  annotations:
    conventions.carto.run/inject-ca-from: "convention-template/webhook-cert"
spec:
  selectorTarget: PodTemplateSpec # optional, defaults to PodTemplateSpec; field optio
ns include PodTemplateSpec|PodIntent
  webhook:
    clientConfig:
      service:
        name: webhook
        namespace: convention-template
```

PodIntent for Cartographer Conventions

This reference topic describes `PodIntent` that you can use with Cartographer Conventions.

Overview

The conditional criteria governing the application of a convention is customizable and is based on the evaluation of a custom Kubernetes resource called PodIntent.

`PodIntent` applies conventions to a workload. A PodIntent is created, or updated, when a workload is run by using a Tanzu Application Platform supply chain.

The `.spec.template`'s PodTemplateSpec is enriched by the conventions and exposed as the `.status.template`s PodTemplateSpec. A log of which sources and conventions are applied is captured with the `conventions.carto.run/applied-conventions` annotation on the PodTemplateSpec.

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  name: sample
spec:
  template:
    spec:
      containers:
      - name: workload
        image: ubuntu
```

BOM for Cartographer Conventions

This reference topic describes the `BOM` structure you can use with Cartographer Conventions.

Overview

The `BOM` is a type/structure wrapping a Software Bill of Materials (SBOM) describing the software components and their dependencies.

Structure

The structure of the `BOM` is defined as follows:

```
{
  "name": "BOM-NAME",
  "raw": "BYTE-ARRAY"
}
```

Where:

- `BOM-NAME` is the prefix `cnb-sbom:`, followed by the location of the BOM definition in the layer for a cloud native buildpack (CNB) SBOM. For example: `cnb-sbom:/layers/sbom/launch/paketo-buildpacks_executable-jar/sbom.cdx.json`. For a non-CNB SBOM, the value of `name` might change.
- `BYTE-ARRAY`: The content of the BOM. The content may be in any format or encoding. Consult the name to infer how the content is structured.

The convention controller forwards BOMs to the convention servers that it can discover from known sources, including:

- CNB-SBOM

Overview of cert-manager

cert-manager adds certificates and certificate issuers as resource types to Kubernetes clusters. It also helps you to obtain, renew, and use those certificates. For more information about cert-manager, see cert-manager documentation.

The cert-manager package allows you to, optionally, configure a number of `ClusterIssuer`. When you install Tanzu Application Platform by using profiles, a self-signed `ClusterIssuer` is included by default.

As of `cert-manager.tanzu.vmware.com/2.0.0`, versioning departs from the upstream, open-source project's version. The contained cert-manager version is reflected in `Package.spec.includedSoftware`. You can identify the version of cert-manager as follows:

```
kubectl get package -n tap-install cert-manager.tanzu.vmware.com.2.0.0 -ojsonpath='{.s
pec.includedSoftware}' | jq
[
 {
   "description": "X.509 certificate management for Kubernetes and OpenShift",
   "displayName": "cert-manager",
   "version": "1.9.1"
 }
]
```


Caution

ACME HTTP01 challenges can fail under certain conditions. For more information, see ACME challenges.

Overview of cert-manager

cert-manager adds certificates and certificate issuers as resource types to Kubernetes clusters. It also helps you to obtain, renew, and use those certificates. For more information about cert-manager, see cert-manager documentation.

The cert-manager package allows you to, optionally, configure a number of `ClusterIssuer`. When you install Tanzu Application Platform by using profiles, a self-signed `ClusterIssuer` is included by default.

As of `cert-manager.tanzu.vmware.com/2.0.0`, versioning departs from the upstream, open-source project's version. The contained cert-manager version is reflected in `Package.spec.includedSoftware`. You can identify the version of cert-manager as follows:

```
kubectl get package -n tap-install cert-manager.tanzu.vmware.com.2.0.0 -ojsonpath='{.s
pec.includedSoftware}' | jq
[
 {
   "description": "X.509 certificate management for Kubernetes and OpenShift",
   "displayName": "cert-manager",
   "version": "1.9.1"
 }
]
```


Caution

ACME HTTP01 challenges can fail under certain conditions. For more information, see ACME challenges.

Install cert-manager

This topic tells you how to install cert-manager from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install cert-manager. For more information about profiles, see Components and installation profiles.

The cert-manager package installs cert-manager and, optionally, a number of `ClusterIssuer`.

To install cert-manager with a self-signed `ClusterIssuer` from the Tanzu Application Platform package repository:

1. List version information for the package by running:

   ```
   tanzu package available list cert-manager.tanzu.vmware.com -n tap-install
   ```

   For example:

   ```
   $ tanzu package available list cert-manager.tanzu.vmware.com -n tap-install
   / Retrieving package versions for cert-manager.tanzu.vmware.com...
     NAME                           VERSION           RELEASED-AT
     cert-manager.tanzu.vmware.com  2.0.0             ...
   ```

2. Discover available configuration for the package by running:

   ```
   tanzu package available get cert-manager.tanzu.vmware.com/2.0.0 --namespace tap
   -install --values-schema
   ```

   For example:

   ```
   $ tanzu package available get cert-manager.tanzu.vmware.com/2.0.0 --namespace t
   ap-install --values-schema

   KEY                    DEFAULT  TYPE    DESCRIPTION
   certManager.pspNames   []       array   PodSecurityPolicy names which cert-manag
   er is allowed to use
   issuers                []       array   The ClusterIssuers to install - default:
   []
   namespace                       string  Cert-manager's namespace - also used as
   its cluster resource namespace
   https://cert-manager.io/v1.9-docs/faq/cluster-resource/
   ```

3. Create a file named `cert-manager-rbac.yaml` by using the following sample:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
     name: cert-manager-tap-install-cluster-admin-role
   rules:
   - apiGroups:
     - '*'
     resources:
     - '*'
     verbs:
     - '*'
   ---
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: cert-manager-tap-install-cluster-admin-role-binding
   roleRef:
     apiGroup: rbac.authorization.k8s.io
   ```

```
  kind: ClusterRole
  name: cert-manager-tap-install-cluster-admin-role
subjects:
- kind: ServiceAccount
  name: cert-manager-tap-install-sa
  namespace: tap-install
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cert-manager-tap-install-sa
  namespace: tap-install
```

Apply the configuration:

```
kubectl apply -f cert-manager-rbac.yaml
```

4. Create a file named `cert-manager-install.yaml` by using the following sample:

```
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: cert-manager
  namespace: tap-install
spec:
  serviceAccountName: cert-manager-tap-install-sa
  packageRef:
    refName: cert-manager.tanzu.vmware.com
    versionSelection:
      constraints: "VERSION-NUMBER"
      prereleases: {}
  values:
    - secretRef:
        name: cert-manager-values

---
apiVersion: v1
kind: Secret
metadata:
  name: cert-manager-values
  namespace: tap-install
stringData:
  values.yaml: |
    issuers:
      - name: tap-ingress-selfsigned
        self_signed: {}
```

Where:

- `VERSION-NUMBER` is the version of the package listed earlier.
- Secret `cert-manager-values` contains your configuration of the cert-manager package.

Apply the configuration:

```
kubectl apply -f cert-manager-install.yaml
```

5. Verify the package installation:

```
tanzu package installed get cert-manager -n tap-install
```

For example:

```
$ tanzu package installed get cert-manager -n tap-install
/ Retrieving installation details for cert-manager...
NAME:                    cert-manager
PACKAGE-NAME:            cert-manager.tanzu.vmware.com
PACKAGE-VERSION:         2.0.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True}]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

6. Verify that cert-manager is up and running:

```
kubectl get deployment -n cert-manager
```

For example:

```
$ kubectl get deployment -n cert-manager
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
cert-manager              1/1     1            1           5m
cert-manager-cainjector   1/1     1            1           5m
cert-manager-webhook      1/1     1            1           5m
```

7. Verify that the self-signed `ClusterIssuer` is present:

```
kubectl get clusterissuer
```

For example:

```
$ kubectl get clusterissuer
NAME                               READY   AGE
tap-ingress-selfsigned             True    5m
tap-ingress-selfsigned-bootstrap   True    5m
...
```

ACME challenges

cert-manager.io provides APIs for managing certificates on Kubernetes. It is one of the most popular extensions for Kubernetes and has found ubiquitous adoption. Automatic Certificate Management Environment (commonly called ACME) is a protocol for automatically obtaining certificates from certificate authorities. LetsEncrypt has designed and pioneered ACME and is one of the most-popular ACME-style, public CA.

You can use ACME with either an HTTP01 or a DNS01 challenge. In both cases the certificate requester must prove ownership of the domain either by answering a plain HTTP request or by setting a DNS record.

When using cert-manager's `(Cluster)Issuer` with an ACME HTTP01 challenge solver, a `Pod` is run and exposed to the network by using ingress. The `Pod` receives the challenge from the CA and responds. If the challenge is solved successfully, the certificate is issued.

When using cert-manager's `(Cluster)Issuer` with an ACME DNS01 challenge solver, the owner of the domain answers the challenge by setting a `TXT` record under the domain name. If the challenge is solved successfully, the certificate is issued.

Working with DNS01 challenges is harder than with HTTP01 challenges, but can work in situations where HTTP01 can't.

HTTP01 challenges can fail

All of components' images of Tanzu Application Platform are relocated to and pulled from a private registry. This also applies to `cert-manager.tanzu.vmware.com`, including the ACME HTTP01 solver `Pod`'s image.

Due to the design of cert-manager's `(Cluster)Issuer` resources, it is not easy to provide them with credentials to your private registry in a way that works consistently across all namespaces in your cluster.

You can deeply integrate a cluster with a private registry so that image pull secrets don't have to be provided explicitely. This is a common practice with popular cloud-based Kubernetes providers such as GKE, AKS and EKS.

As a result, ACME HTTP01 challenges can fail when your cluster is not deeply integrated with your private registry. In that case VMware recommends the following workarounds:

- (Recommended) Use DNS01 challenges
- Provide your `(Cluster)Issuer` with a `ServiceAccount` by using its pod template so that it can pull from your registry. For example:

  ```
  ---
  apiVersion: v1
  kind: ServiceAccount
  metadata:
   name: tap-acme-http01-solver
   namespace: #! ...
  imagePullSecrets:
   - registry-credentials

  ---
  apiVersion: cert-manager.io/v1
  kind: ClusterIssuer
  metadata: #! ...
  spec:
   #! ...
   acme:
     solvers:
       - http01:
           ingress:
             podTemplate:
               spec:
                 serviceAccountName: tap-acme-http01-solver
  ```

  The challenge lies in ensuring that the same `ServiceAccount` is available in every namespace that obtains `Certificates` from the `ClusterIssuer`.

Overview of Cloud Native Runtimes

Cloud Native Runtimes for Tanzu is a serverless application runtime for Kubernetes that is based on Knative and runs on a single Kubernetes cluster.

To learn more about Cloud Native Runtimes, see Cloud Native Runtimes for VMware Tanzu.

Overview of Cloud Native Runtimes

Cloud Native Runtimes for Tanzu is a serverless application runtime for Kubernetes that is based on Knative and runs on a single Kubernetes cluster.

To learn more about Cloud Native Runtimes, see Cloud Native Runtimes for VMware Tanzu.

Install Cloud Native Runtimes

This topic describes how you can install Cloud Native Runtimes from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Cloud Native Runtimes. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Cloud Native Runtimes:

- Complete all prerequisites to install Tanzu Application Platform. See Prerequisites.
- Ensure that Contour v1.22.0 or later is installed. Tanzu Application Platform includes a correctly versioned package of Contour if you do not have it installed already.

Install

To install Cloud Native Runtimes:

1. List version information for the package by running:

   ```
   tanzu package available list cnrs.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list cnrs.tanzu.vmware.com --namespace tap-install
   - Retrieving package versions for cnrs.tanzu.vmware.com...
     NAME                   VERSION  RELEASED-AT
     cnrs.tanzu.vmware.com  2.2.0    2023-03-08 16:00:00 -0800 PST
   ```

2. (Optional) Make changes to the default installation settings:

   1. Gather values schema.

      ```
      tanzu package available get cnrs.tanzu.vmware.com/2.2.0 --values-schema -
      n tap-install
      ```

      For example:

      ```
      $ tanzu package available get cnrs.tanzu.vmware.com/2.2.0 --values-schema
      -n tap-install
      | Retrieving package details for cnrs.tanzu.vmware.com/2.2.0...
        KEY                          DEFAULT                                TYPE
      DESCRIPTION
        https_redirection            true                                   boole
      an  CNRs ingress will send a 301 redirect for all http connections, askin
      g the clients to use HTTPS
        ingress.internal.namespace   tanzu-system-ingress                    strin
      g   Required. Specify a namespace where an existing Contour is installed
      on your cluster. CNR will use this Contour instance for internal service
      s.
        ingress.external.namespace   tanzu-system-ingress                    strin
      g   Required. Specify a namespace where an existing Contour is installed
      on your cluster. CNR will use this Contour instance for external service
      s.
        ingress_issuer                                                      strin
      g   Cluster issuer to be used in CNRs. To use this property the domain_na
      ```

```
me or domain_config must be set. Under the hood, when this property is se
t auto-tls is Enabled.
  namespace_selector                                                strin
g   Specifies a LabelSelector which determines which namespaces should ha
ve a wildcard certificate provisioned. Set this property only if the Clus
ter issuer is type DNS-01 challenge.
  domain_config               <nil>                                 <nil>
Optional. Overrides the Knative Serving "config-domain" ConfigMap, allowi
ng you to map Knative Services to specific domains. Must be valid YAML an
d conform to the "config-domain" specification.
  domain_template             {{.Name}}.{{.Namespace}}.{{.Domain}}  strin
g   Optional. Specifies the golang text template string to use when const
ructing the DNS name for a Knative Service.
  lite.enable                 false                                 <nil>
Optional. Set to "true" to enable lite mode. Reduces CPU and Memory resou
rce requests for all cnrs Deployments, Daemonsets, and StatefulSets by ha
lf. Not recommended for production.
  pdb.enable                  true                                  <nil>
Optional. Set to true to enable a PodDisruptionBudget for the Knative Ser
ving activator and webhook deployments.
  default_tls_secret                                                strin
g   Optional. Specify a fallback TLS Certificate for use by Knative Servi
ces if autoTLS is disabled. Will set default exterenal scheme for Knative
Service URLs to "https". Requires either "domain_name" or "domain_config"
to be set.
  kubernetes_version          0.0.0                                 <nil>
Optional. Version of K8s infrastructure being used. Supported Values: val
id Kubernetes major.minor.patch versions
  ca_cert_data                                                      strin
g   Optional. PEM Encoded certificate data to trust TLS connections with
a private CA.
  provider                    <nil>                                 <nil>
Deprecated. Instead, use "lite.enable" and "pdb.enable" options combined.
Supported Values: local
  domain_name                                                       strin
g   Optional. Default domain name for Knative Services.
  kubernetes_distribution     <nil>                                 <nil>
Optional. Type of K8s infrastructure being used. Supported Values: opensh
ift
```

2. Create a `cnr-values.yaml` by using the following sample as a guide:

```
---
domain_name: example.com
ingress:
external:
    namespace: tanzu-system-ingress
internal:
    namespace: tanzu-system-ingress
```

Note

For most installations, you can leave the `cnr-values.yaml` empty, and use the default values.

If you are running on a single-node cluster, such as kind or minikube, set the `lite.enable: true` option. This option reduces resources requests for Cloud Native Runtimes deployments.

Cloud Native Runtimes uses the existing Contour installation in the `tanzu-system-ingress` namespace by default for external and internal access.

If your environment has Contour installed already, and it is not the Tanzu Application Platform provided Contour, you can configure Cloud Native Runtimes to use it. See Installing Cloud Native Runtimes for Tanzu with an Existing Contour Installation in the Cloud Native Runtimes documentation.

3. Install the package by running:

```
tanzu package install cloud-native-runtimes \
  --package cnrs.tanzu.vmware.com \
  --version 2.2.0 \
  --namespace tap-install \
  --values-file cnr-values.yaml \
  --poll-timeout 30m
```

For example:

```
$ tanzu package install cloud-native-runtimes \
    --package cnrs.tanzu.vmware.com \
    --version 2.2.0 \
    --namespace tap-install \
    --values-file cnr-values.yaml \
    --poll-timeout 30m

| Installing package 'cnrs.tanzu.vmware.com'
| Getting package metadata for 'cnrs.tanzu.vmware.com'
| Creating service account 'cloud-native-runtimes-tap-install-sa'
| Creating cluster admin role 'cloud-native-runtimes-tap-install-cluster-role'
| Creating cluster role binding 'cloud-native-runtimes-tap-install-cluster-role
binding'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'cloud-native-runtimes' in namespace 'tap-install'
```

Use an empty file for `cnr-values.yaml` if you want the default installation configuration. Otherwise, see the earlier step to learn more about setting installation configuration values.

4. Verify the package install by running:

```
tanzu package installed get cloud-native-runtimes -n tap-install
```

For example:

```
tanzu package installed get cloud-native-runtimes -n tap-install
| Retrieving installation details for cc...
NAME:                    cloud-native-runtimes
PACKAGE-NAME:            cnrs.tanzu.vmware.com
PACKAGE-VERSION:         2.2.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

5. Configure a namespace to use Cloud Native Runtimes:

Important

This step covers configuring a namespace to run Knative services. If you rely on a SupplyChain to deploy Knative services into your cluster, skip this step because namespace configuration is covered in Set up developer

namespaces to use your installed packages. Otherwise, you must follow these steps for each namespace where you create Knative services.

Service accounts that run workloads using Cloud Native Runtimes need access to the image pull secrets for the Tanzu package. This includes the `default` service account in a namespace, which is created automatically but not associated with any image pull secrets. Without these credentials, attempts to start a service fail with a timeout and the pods report that they are unable to pull the `queue-proxy` image.

1. Create an image pull secret in the current namespace and fill it from the `tap-registry` secret mentioned in Add the Tanzu Application Platform package repository. Run these commands to create an empty secret and annotate it as a target of the secretgen controller:

   ```
   kubectl create secret generic pull-secret --from-literal=.dockerconfigjson={} --type=kubernetes.io/dockerconfigjson
   kubectl annotate secret pull-secret secretgen.carvel.dev/image-pull-secret=""
   ```

2. After you create a `pull-secret` secret in the same namespace as the service account, run the following command to add the secret to the service account:

   ```
   kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "pull-secret"}]}'
   ```

3. Verify that a service account is correctly configured by running:

   ```
   kubectl describe serviceaccount default
   ```

   For example:

   ```
   kubectl describe sa default
   Name:                default
   Namespace:           default
   Labels:              <none>
   Annotations:         <none>
   Image pull secrets:  pull-secret
   Mountable secrets:   default-token-xh6p4
   Tokens:              default-token-xh6p4
   Events:              <none>
   ```

   Note

   The service account has access to the `pull-secret` image pull secret.

Overview of Contour

Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multi-team ingress delegation. It provides the control plane for the Envoy edge and service proxy. For more information about Contour, see Contour documentation.

Overview of Contour

Contour is an ingress controller for Kubernetes that supports dynamic configuration updates and multi-team ingress delegation. It provides the control plane for the Envoy edge and service proxy. For more information about Contour, see Contour documentation.

Install Contour

This topic tells you how to install Contour from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Contour. For more information about profiles, see Components and installation profiles.

To install Contour from the Tanzu Application Platform package repository without a profile:

1. Install cert-manager.
2. List version information for the package by running:

   ```
   tanzu package available list contour.tanzu.vmware.com -n tap-install
   ```

   For example:

   ```
   $  tanzu package available list contour.tanzu.vmware.com -n tap-install
   - Retrieving package versions for contour.tanzu.vmware.com...
     NAME                       VERSION       RELEASED-AT
     contour.tanzu.vmware.com  1.22.0+tap.8  2022-12-05 19:00:00 -0500 EST
   ```

3. Create a file named `contour-rbac.yaml` by using the following sample and apply the configuration:

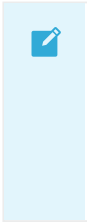
   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRole
   metadata:
     name: contour-tap-install-cluster-admin-role
   rules:
   - apiGroups:
     - '*'
     resources:
     - '*'
     verbs:
     - '*'
   ---
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: contour-tap-install-cluster-admin-role-binding
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: contour-tap-install-cluster-admin-role
   subjects:
   - kind: ServiceAccount
     name: contour-tap-install-sa
     namespace: tap-install
   ---
   apiVersion: v1
   kind: ServiceAccount
   metadata:
   ```

```
  name: contour-tap-install-sa
  namespace: tap-install
```

4. Apply the configuration by running:

```
kubectl apply -f contour-rbac.yaml
```

5. Create a file named `contour-install.yaml` by using the following sample and apply the configuration:



Note

The following configuration installs the Contour package with default options. To make changes to the default installation settings, go to the next step.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: contour
  namespace: tap-install
spec:
  serviceAccountName: contour-tap-install-sa
  packageRef:
    refName: contour.tanzu.vmware.com
    versionSelection:
      constraints: "VERSION-NUMBER"
      prereleases: {}
```

Where `VERSION-NUMBER` is the version of the package listed earlier.

6. (Optional) Make changes to the default installation settings:

    1. Gather values schema by running:

    ```
    tanzu package available get contour.tanzu.vmware.com/1.22.0+tap.8 --value
    s-schema -n tap-install
    ```

    For example:

    ```
    $ tanzu package available get contour.tanzu.vmware.com/1.22.0+tap.8 --val
    ues-schema -n tap-install

      KEY                                DEFAULT              TYPE     DES
    CRIPTION
      kubernetes_version                 0.0.0                string   Opt
    ional. Version of K8s infrastructure being used. Supported Values: valid
    Kubernetes major.minor.patch versions
      namespace                          tanzu-system-ingress  string   The
    namespace in which to deploy Contour and Envoy.
      certificates.duration              8760h                string   If
    using cert-manager, how long the certificates should be valid for. If use
    CertManager is false, this field is ignored.
      certificates.renewBefore           360h                 string   If
    using cert-manager, how long before expiration the certificates should be
    renewed. If useCertManager is false, this field is ignored.
      contour.configFileContents         <nil>                <nil>    The
    YAML contents of the Contour config file. See https://projectcontour.io/d
    ocs/v1.22.0/configuration/#configuration-file for more information.
      contour.logLevel                   info                 string   The
    Contour log level. Valid options are 'info' and 'debug'.
      contour.replicas                   2                    integer  How
    ```

```
many Contour pod replicas to have.
  contour.useProxyProtocol             false                 boolean  Whe
ther to enable PROXY protocol for all Envoy listeners.
  envoy.terminationGracePeriodSeconds  300                   integer  The
termination grace period, in seconds, for the Envoy pods.
  envoy.workload.replicas              2                     integer  The
number of Envoy replicas to deploy when 'type' is set to 'Deployment'. If
not specified, it will default to '2'.
  envoy.workload.type                  DaemonSet             string   The
type of Kubernetes workload Envoy is deployed as. Options are 'Deploymen
t' or 'DaemonSet'. If not specified, will default to 'DaemonSet'.
  envoy.hostNetwork                    false                 boolean  Whe
ther to enable host networking for the Envoy pods.
  envoy.hostPorts.enable               true                  boolean  Whe
ther to enable host ports. If false, http & https are ignored.
  envoy.hostPorts.http                 80                    integer  If
enable == true, the host port number to expose Envoy's HTTP listener on.
  envoy.hostPorts.https                443                   integer  If
enable == true, the host port number to expose Envoy's HTTPS listener on.
  envoy.logLevel                       info                  string   The
Envoy log level.
  envoy.service.nodePorts.https        0                     integer  The
node port number to expose Envoy's HTTPS listener on. If not specified, a
node port will be auto-assigned by Kubernetes.
  envoy.service.nodePorts.http         0                     integer  The
node port number to expose Envoy's HTTP listener on. If not specified, a
node port will be auto-assigned by Kubernetes.
  envoy.service.type                                         string   The
type of Kubernetes service to provision for Envoy. If not specified, will
default to 'NodePort' for docker and vsphere and 'LoadBalancer' for other
s.
  envoy.service.annotations            <nil>                 <nil>    Ann
otations to set on the Envoy service.
  envoy.service.aws.LBType             classic               string   AWS
loadbalancer type.
  envoy.service.externalTrafficPolicy                        string   The
external traffic policy for the Envoy service. If type is 'ClusterIP', th
is field is ignored. Otherwise, defaults to 'Cluster' for vsphere and 'Lo
cal' for others.
  envoy.service.loadBalancerIP                               string   The
desired load balancer IP. If type is not 'LoadBalancer', this field is ig
nored. It is up to the cloud provider whether to honor this request. If n
ot specified, then load balancer IP will be assigned by the cloud provide
r.
  infrastructure_provider              vsphere               string   The
underlying infrastructure provider. Valid values are `vsphere`, `aws` and
`azure`.
  kubernetes_distribution                                    string   Kub
ernetes distribution that this package is being installed on. Supported v
alues: ['','openshift']
```

2. Create a `contour-install.yaml` file by using the following sample as a guide:

Note

This sample is for installation in an AWS public cloud with `LoadBalancer` services.

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: contour
  namespace: tap-install
```

```
spec:
  serviceAccountName: contour-tap-install-sa
  packageRef:
    refName: contour.tanzu.vmware.com
    versionSelection:
      constraints: 1.22.0+tap.8
      prereleases: {}
  values:
  - secretRef:
      name: contour-values
---
apiVersion: v1
kind: Secret
metadata:
  name: contour-values
  namespace: tap-install
stringData:
  values.yaml: |
    envoy:
      service:
        type: LoadBalancer
    infrastructure_provider: aws
```

The LoadBalancer type is appropriate for most installations, but local clusters such as `kind` or `minikube` can fail to complete the package install if LoadBalancer services are not supported.

For local clusters, you can configure `contour.envoy.service.type` to be `NodePort`. If your local cluster is set up with extra port mappings on the nodes, you might also need configure `envoy.service.nodePorts.http` and `envoy.service.nodePorts.https` to match the port mappings from your local machine into one of the nodes of your local cluster. This pattern is seen when using the Learning Center on Kind.

Contour provides an Ingress implementation by default. If you have another Ingress implementation in your cluster, you must explicitly specify an IngressClass to select a particular implementation.

Cloud Native Runtimes programs Contour HTTPRoutes are based on the installed namespace. The default installation of CNR uses a single Contour to provide internet-visible services. You can install a second Contour instance with service type `ClusterIP` if you want to expose some services to only the local cluster. The second instance must be installed in a separate namespace. You must set the CNR value `ingress.internal.namespace` to point to this namespace.

7. Install the package by running:

```
kubectl apply -f contour-install.yaml
```

8. Verify the package install by running:

```
tanzu package installed get contour -n tap-install
```

For example:

```
$ tanzu package installed get contour -n tap-install
/ Retrieving installation details for contour...
NAME:                    contour
PACKAGE-NAME:            contour.tanzu.vmware.com
PACKAGE-VERSION:         1.22.0+tap.8
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

9. Verify the installation by running:

```
kubectl get po -n tanzu-system-ingress
```

For example:

```
$  kubectl get po -n tanzu-system-ingress
NAME                          READY   STATUS    RESTARTS   AGE
contour-857d46c845-4r6c5   1/1     Running   1          18d
contour-857d46c845-p6bbq   1/1     Running   1          18d
envoy-mxkjk                2/2     Running   2          18d
envoy-qlg8l                2/2     Running   2          18d
```

Ensure that all pods are `Running` with all containers ready.

Configure Cipher Suites and TLS version in Contour

This topic tells you how to configure TLS options for Contour in Tanzu Application Platform (commonly known as TAP).

Contour provides configuration options for TLS version and Cipher Suites. Rather than directly exposed through a top level key in the pacakge, they fall into the category of advanced Contour configurations by using the `contour.configFileContents` key. For more information about these configuration options, see Contour documentation.

To configure TLS options for Contour in Tanzu Application Platform, edit the `contour` section of your TAP values file as follows:

```
contour:
  # ... there maybe some configuration already here
  contour:
    configFileContents:
      tls:
        minimum-protocol-version: "1.2"
        cipher-suites:
        - '[ECDHE-ECDSA-AES128-GCM-SHA256|ECDHE-ECDSA-CHACHA20-POLY1305]'
        - '[ECDHE-RSA-AES128-GCM-SHA256|ECDHE-RSA-CHACHA20-POLY1305]'
        - 'ECDHE-ECDSA-AES256-GCM-SHA384'
        - 'ECDHE-RSA-AES256-GCM-SHA384'
```

Expect to see the following Cipher Suites and TLS version data in the Contour configmap:

```
$ kubectl -n tanzu-system-ingress get configmap contour -oyaml
apiVersion: v1
data:
  contour.yaml: |
    tls:
      minimum-protocol-version: "1.2"
      cipher-suites:
      - '[ECDHE-ECDSA-AES128-GCM-SHA256|ECDHE-ECDSA-CHACHA20-POLY1305]'
      - '[ECDHE-RSA-AES128-GCM-SHA256|ECDHE-RSA-CHACHA20-POLY1305]'
      - ECDHE-ECDSA-AES256-GCM-SHA384
      - ECDHE-RSA-AES256-GCM-SHA384
kind: ConfigMap
metadata:
...
```

Important

To update the configmap, you must configure it through Tanzu Application Platform values file. If you change it directly in the configmap, kapp-controller reverts all the changes you made.

Configure Contour

This topic tells you how to configure Contour to best suit your cluster.

By default, Contour installs with the Controllers as a `Deployment` and the Envoys as a `DaemonSet`. In most cases, this is sufficient. However, VMware recommends running Envoy as a `Deployment` in the following scenarios:

- Smaller Clusters
- Larger Clusters

Smaller Clusters

On most clusters, a `DaemonSet` works without any issues. However, if you limit resources per node and the nodes are heavily used, deploying Envoy as a `DaemonSet` might consume unnecessary resources on every node. In this case, VMware recommends using `Deployment` with a fixed number of replicas.

Larger Clusters

On larger clusters, running Envoy as a `DaemonSet` might be inefficient. The more Envoys in the cluster, the more resources the Contour controller needs to keep them updated. If the Contour controllers use lots of resources, VMware recommends running Envoy as a `Deployment`.

Configuring Envoy as a Deployment

To configure Envoy as a `Deployment`, update your Contour values file as follows:

```
envoy:
  workload:
    type: Deployment
    replicas: N
```

If you use a Tanzu Application Platform values file, you can add the these configurations to the `contour` section.

Overview of Crossplane

Crossplane is an open source, Cloud Native Computing Foundation (CNCF) project built on the foundation of Kubernetes. Tanzu Application Platform (commonly known as TAP) uses Crossplane to power a number of capabilities, such as dynamic provisioning of services instances with Services Toolkit and the Bitnami Services.

Crossplane with Tanzu Application Platform

Tanzu Application Platform includes a Carvel package named `crossplane.tanzu.vmware.com`, which is included by default in the full, iterate, and run profiles. The package installs Upbound Universal Crossplane (UXP).

In addition, the package includes two pre-configured Crossplane providers: provider-helm and provider-kubernetes. Both of providers provide useful Managed Resources that you can use as part of Composition. These are both used by Tanzu Application Platform's Bitnami Services.

The package installs UXP and the providers to the `crossplane-system` namespace.

Getting started

To learn about working with Crossplane in general, see the Crossplane documentation. To learn about how Tanzu Application Platform integrates with Crossplane, see one of the following tutorials to get started.

For apps teams:

- Tutorial: Working with Bitnami Services

For ops teams:

- Tutorial: Setup Dynamic Provisioning of Service Instances

Alternatively, see the reference material.

Overview of Crossplane

Crossplane is an open source, Cloud Native Computing Foundation (CNCF) project built on the foundation of Kubernetes. Tanzu Application Platform (commonly known as TAP) uses Crossplane to power a number of capabilities, such as dynamic provisioning of services instances with Services Toolkit and the Bitnami Services.

Crossplane with Tanzu Application Platform

Tanzu Application Platform includes a Carvel package named `crossplane.tanzu.vmware.com`, which is included by default in the full, iterate, and run profiles. The package installs Upbound Universal Crossplane (UXP).

In addition, the package includes two pre-configured Crossplane providers: provider-helm and provider-kubernetes. Both of providers provide useful Managed Resources that you can use as part of Composition. These are both used by Tanzu Application Platform's Bitnami Services.

The package installs UXP and the providers to the `crossplane-system` namespace.

Getting started

To learn about working with Crossplane in general, see the Crossplane documentation. To learn about how Tanzu Application Platform integrates with Crossplane, see one of the following tutorials to get started.

For apps teams:

- Tutorial: Working with Bitnami Services

For ops teams:

- Tutorial: Setup Dynamic Provisioning of Service Instances

Alternatively, see the reference material.

Install Crossplane

This topic tells you how to install Crossplane from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Crossplane. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Crossplane:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install Crossplane

To install Crossplane:

1. See what versions of Crossplane are available to install by running:

   ```
   tanzu package available list -n tap-install crossplane.tanzu.vmware.com
   ```

   For example:

   ```
   $ tanzu package available list -n tap-install crossplane.tanzu.vmware.com
     NAME                              VERSION           RELEASED-AT
     crossplane.tanzu.vmware.com       0.1.1             2023-03-10 14:24:35 +000
   0 UTC
   ```

2. Install Crossplane by running:

   ```
   tanzu package install crossplane \
     --package crossplane.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the Crossplane version you want to install. For example, `0.1.1`.

3. Verify that the package installed by running:

   ```
   tanzu package installed get crossplane -n tap-install
   ```

   In the output, confirm that the `STATUS` value is `Reconcile succeeded`.

   For example:

   ```
   $ tanzu package installed get crossplane -n tap-install
   NAMESPACE:          tap-install
   NAME:               crossplane
   PACKAGE-NAME:       crossplane.tanzu.vmware.com
   PACKAGE-VERSION:    0.1.1
   STATUS:             Reconcile succeeded
   CONDITIONS:         - type: ReconcileSucceeded
     status: "True"
     reason: ""
     message: ""
   ```

Crossplane reference

This section provides reference documentation for Crossplane.

In this section:

Package values for Crossplane

This topic lists the keys and values you can use to configure the behavior of the Crossplane package. Configuration is split between configuration specific to Crossplane in Tanzu Application Platform (commonly known as TAP) and configuration of the Upbound Universal Crossplane (UXP) Helm Chart.

If you are applying configuration to Tanzu Application Platform through the use of profiles and the `tap-values.yaml`, all configuration must exist under the `crossplane` top-level key.

For example:

```
crossplane:
  replicas: 3
```

Tanzu Application Platform configuration

The following table lists configuration specific to Crossplane in Tanzu Application Platform.

KEY	DEFAULT	TYPE	DESCRIPTION
kubernetes_version	`""`	string	Optional: The Kubernetes version. Valid values are `''` or take the form `'1.25.0'`
kubernetes_distribution	`""`	string	Optional: The Kubernetes distribution. Valid values are `''` or `'openshift'`

Standard Crossplane configuration

The following table lists configuration for the UXP Helm Chart.

KEY	DEFAULT	TYPE	DESCRIPTION
replicas	`1`	integer	
serviceAccount.customAnnotations	`'{}'`	object	
bootstrapper.resources	`'{}'`	object	
bootstrapper.config.args		array	
bootstrapper.config.debugMode	`false`	boolean	
bootstrapper.config.envVars	`'{}'`	object	
bootstrapper.image.pullPolicy	`IfNotPresent`	string	

KEY	DEFAULT	TYPE	DESCRIPTION
bootstrapper.image.repository	`xpkg.upbound.io/upbound/uxp-bootstrapper`	string	
bootstrapper.image.tag	`v1.11.0-up.1`	string	
maxReconcileRate	`""`	string	How frequently Crossplane reconciles its resources in seconds.
metrics.enabled	`false`	boolean	
securityContextCrossplane.allowPrivilegeEscalation	`false`	boolean	
securityContextCrossplane.readOnlyRootFilesystem	`true`	boolean	
securityContextCrossplane.runAsGroup	`65532`	integer	
securityContextCrossplane.runAsUser	`65532`	integer	
xfn.securityContext.allowPrivilegeEscalation	`false`	boolean	
xfn.securityContext.capabilities.add		array	
xfn.securityContext.readOnlyRootFilesystem	`true`	boolean	
xfn.securityContext.runAsGroup	`65532`	integer	
xfn.securityContext.runAsUser	`65532`	integer	
xfn.securityContext.seccompProfile.type	`Unconfined`	string	
xfn.args	`'{}'`	object	
xfn.cache.configMap	`""`	string	
xfn.cache.medium	`""`	string	
xfn.cache.pvc	`""`	string	
xfn.cache.sizeLimit	`1Gi`	string	

KEY	DEFAULT	TYPE	DESCRIPTION
xfn.enabled	`false`	boolean	
xfn.extraEnvVars	`'{}'`	object	
xfn.image.pullPolicy	`IfNotPresent`	string	
xfn.image.repository	`crossplane/xfn`	string	
xfn.image.tag	`v1.11.0-up.1`	string	
xfn.resources.requests.cpu	`1000m`	string	
xfn.resources.requests.memory	`1Gi`	string	
xfn.resources.limits.memory	`2Gi`	string	
xfn.resources.limits.cpu	`2000m`	string	
resourcesCrossplane.limits.cpu	`500m`	string	
resourcesCrossplane.limits.memory	`1Gi`	string	
resourcesCrossplane.requests.cpu	`250m`	string	
resourcesCrossplane.requests.memory	`768Mi`	string	
resourcesRBACManager.limits.memory	`768Mi`	string	
resourcesRBACManager.limits.cpu	`100m`	string	
resourcesRBACManager.requests.cpu	`100m`	string	
resourcesRBACManager.requests.memory	`512Mi`	string	
configuration.packages		array	
deploymentStrategy	`RollingUpdate`	string	
extraEnvVarsCrossplane	`'{}'`	object	List of extra environment variables to set in the Crossplane deployment. For example, `extraEnvironmentVars: sample.key: value1 ANOTHER.KEY: value2` results with `- name: sample_key value: "value1" - name: ANOTHER_KEY value: "value2"`

KEY	DEFAULT	TYPE	DESCRIPTION
registryCaBundleConfig	`'{}'`	object	
billing.awsMarketplace.iamRoleARN	`arn:aws:iam::ACCOUNT-ID>:role/ROLE-NAME`	string	
billing.awsMarketplace.enabled	`false`	boolean	
image.pullPolicy	`IfNotPresent`	string	
image.repository	`upbound/crossplane`	string	
image.tag	`v1.11.0-up.1`	string	
packageCache.configMap	`""`	string	
packageCache.medium	`""`	string	
packageCache.pvc	`""`	string	
packageCache.sizeLimit	`5Mi`	string	
securityContextRBACManager.allowPrivilegeEscalation	`false`	boolean	
securityContextRBACManager.readOnlyRootFilesystem	`true`	boolean	
securityContextRBACManager.runAsGroup	`65532`	integer	
securityContextRBACManager.runAsUser	`65532`	integer	
rbacManager.managementPolicy	`Basic`	string	
rbacManager.nodeSelector	`'{}'`	object	
rbacManager.affinity	`'{}'`	object	
rbacManager.args	`'{}'`	object	
rbacManager.deploy	`true`	boolean	

KEY	DEFAULT	TYPE	DESCRIPTION
rbacManager.leaderElection	`true`	boolean	
rbacManager.replicas	`1`	integer	
rbacManager.skipAggregatedClusterRoles	`true`	boolean	
rbacManager.tolerations		array	
customAnnotations	`'{}'`	object	Custom annotations to add to the Crossplane deployment and pod.
customLabels	`'{}'`	object	Custom labels to add into metadata.
leaderElection	`true`	boolean	
nodeSelector	`'{}'`	object	
podSecurityContextCrossplane	`'{}'`	object	
webhooks.enabled	`false`	boolean	
args	`'{}'`	object	
podSecurityContextRBACManager	`'{}'`	object	
provider.packages		array	
tolerations		array	
upbound.controlPlane.permission	`edit`	string	
affinity	`'{}'`	object	
extraEnvVarsRBACManager	`'{}'`	object	List of extra environment variables to set in the Crossplane RBAC manager deployment. For example, `extraEnvironmentVars: sample.key: value1 ANOTHER.KEY: value2` results with `- name: sample_key value: "value1" - name: ANOTHER_KEY value: "value2"`
nameOverride	`crossplane`	string	
priorityClassName	`""`	string	

Version matrix for Crossplane

This topic provides you with a version matrix for the Crossplane package and its open source components in Tanzu Application Platform v1.5 (commonly known as TAP).

To view this information for another Tanzu Application Platform version, select the version from the drop-down menu at the top of this page.

The following table has the component versions for the Crossplane package.

Component	Version
Crossplane package	0.1.1
UXP	1.11.0-up.1
Upbound Crossplane	1.11.1-up.1
provider-helm	0.14.0
provider-kubernetes	commit SHA 725baeed

Note

Tanzu Application Platform patch releases are only added to the table when there is a change to one or more of the other versions in the table. Otherwise, the corresponding versions remain the same for each Tanzu Application Platform patch release.

Crossplane limitations

This topic tells you about the limitations related to Crossplane on Tanzu Application Platform (commonly known as TAP).

For troubleshooting guidance, see Troubleshoot Crossplane.

Cluster performance degradation due to large number of CRDs

Take care before installing extra Crossplane Providers into Tanzu Application Platform. Some Providers install hundreds of additional CRDs into the cluster.

This is particularly true of the Providers for AWS, Azure, and GCP. For the number of CRDs installed with these Providers, see:

- provider-aws CRDs
- provider-azure CRDs
- provider-gcp CRDs

You must ensure that your cluster has sufficient resource to support this number of additional CRDs if you choose to install them.

Troubleshoot Crossplane

This topic explains how you troubleshoot issues related to Crossplane on Tanzu Application Platform (commonly known as TAP).

For the limitations of Crossplane, see Crossplane limitations.

Crossplane Providers do not transition to HEALTHY=True if using a custom certificate for your registry

Symptom:

A known issue occurs when you install and configure Crossplane Providers while using a custom certificate for your registry. The issue prevents the class claims used for dynamic provisioning from reconciling. A common symptom of this issue is that class claims indefinitely report a status condition of `ResourceMatched=False` with `Reason=ResourceNotReady`.

You can confirm the current status of the Crossplane Providers by running:

```
kubectl get providers
```

Example output:

```
NAME                  INSTALLED   HEALTHY   PACKAGE
AGE
provider-helm                             registry.example.com/tap/tap-packages:prov
ider-helm@sha256:a3a14b07b79a8983257d1a2cc0449a4806753868178055554cfa38de7b649467
3d5h
provider-kubernetes                       registry.example.com/tap/tap-packages:prov
ider-kubernetes@sha256:8039f7e56376b46532e3ce0eb7fc4a4501f2d85decf4912bb5952083abb41b7
b   3d5h
```

In this example, the Providers are not reporting `INSTALLED=True` or `HEALTHY=True`. Therefore, they might be affected by this issue.

Cause:

This issue occurs because the Providers are installed by Crossplane itself rather than directly by the Tanzu Application Platform Crossplane Package. CA certificate data configured through the `tap-values.yaml` file is not passed down to Crossplane, and therefore it is unable to pull the Provider images.

Solution:

Create a `ConfigMap` with the PEM encoded certificate data for your CA certificate, and then set `crossplane.registryCaBundleConfig` to refer to the `ConfigMap` in `tap-values.yaml`.

Note

From Tanzu Application Platform v1.6, the Crossplane Package inherits the data configured in `shared.ca_cert_data` of `tap-values.yaml` and this workaround will no longer be needed.

Crossplane Providers cannot communicate with systems using a custom CA

Symptom:

A known issue occurs when a Crossplane Provider needs to communicate with systems that are set up with a custom CA. For example, when the Crossplane Helm Provider uses `releases.helm.crossplane.io` to try to pull a Helm chart from a registry that uses a custom CA, you see that:

- The `releases.helm.crossplane.io` never reports the status condition `Ready=True`.
- The `releases.helm.crossplane.io` shows a certificate verification error either:
  - Under the status condition of type `SYNCED`.
  - For the event on the `release.helm.crossplane.io`.

To confirm whether you are affected by this issue:

1. Verify the status by running:

   ```
   kubectl get releases.helm.crossplane.io
   ```

   Example output if you are affected by the issue:

   ```
   NAME                CHART   VERSION   SYNCED   READY   STATE   REVISION   DESCR
   IPTION   AGE
   wordpress-example           15.2.5    False    False
   7m37s
   ```

2. Find out more about the reason by running the following command, or similar:

   ```
   kubectl get event --namespace default --field-selector involvedObject.name=word
   press-example,involvedObject.kind=Release,type!=Normal | grep -e 'LAST SEEN' -e
   'failed to login'
   ```

   Example output if you are affected by the issue:

   ```
   LAST SEEN   TYPE      REASON                            OBJECT
   MESSAGE
   3m41s       Warning   CannotCreateExternalResource      release/wordpress-examp
   le   failed to install release: failed to login to registry: Get "https://insec
   ure-registry:443/v2/": tls: failed to verify certificate: x509: certificate sig
   ned by unknown authority
   ```

Cause:

This issue occurs because the Providers are installed by Crossplane itself rather than directly by the Tanzu Application Platform Crossplane Package. CA certificate data configured through the `tap-values.yaml` file is not passed down to Crossplane Providers. Therefore, the Providers are unable to connect to the systems they need to communicate with, such as, the Helm Provider connecting to a registry hosting the Helm charts or the Kubernetes Provider connecting to a Kubernetes APIServer. This issue might be resolved in a later release.

Solution:

Use admission control that allows you to mutate objects, in this case pods, and injects the appropriate CA certificates. You can use any system that can mutate objects at admission to mutate the Crossplane Provider pods. For example, to inject CA certificates you can use this sample in the Kyverno documentation.

Developer Conventions overview

Developer Conventions is a set of conventions that enable your workloads to support live-update and debug operations in Tanzu Application Platform (commonly known as TAP).

Prerequisites

- Tanzu CLI Apps plug-in
- Tanzu Dev Tools for VSCode IDE extension.

Features

Enabling Live Updates

Developer Conventions modifies your workload to enable live updates in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--live-update=true`. For more information about how to deploy a workload with the CLI, see Tanzu apps workload apply.
- You deploy a workload by using the `Tanzu: Live Update Start` option through the Tanzu Developer Tools for VS Code extension. For more information about live updating with the extension, see Overview of Tanzu Developer Tools for Visual Studio Code.

When either of the preceding actions take place, the convention behaves as follows:

1. Looks for the `apps.tanzu.vmware.com/live-update=true` annotation on a PodTemplateSpec associated with a workload.
2. Verifies that the image to which conventions are applied contains a process that can be live updated.
3. Adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod is not scaled down to 0 during a live update session.

After these changes are made, you can use the Tanzu Dev Tools extension or the Tilt CLI to make live update changes to source code directly on the cluster.

Enabling debugging

Developer Conventions modifies your workload to enable debugging in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--debug=true`. For more information about how to deploy a workload with the CLI, see Tanzu apps workload apply.
- You deploy a workload by using the `Tanzu Java Debug Start` option through the Tanzu Developer Tools for VS Code extension. For more information about debugging with the extension, see Overview of Tanzu Developer Tools for Visual Studio Code.

When either of the preceding actions take place, the convention behaves as follows:

1. It looks for the `apps.tanzu.vmware.com/debug=true` annotation on a PodTemplateSpec associated with a workload.
2. It checks for the `debug-8` or `debug-9` labels on the image configuration's bill of materials (BOM).
3. It sets the TimeoutSeconds of the Liveness, Readiness, and Startup probes to 600 if currently set to a lower number.
4. It adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod won't be scaled down to 0 during a debug session.

After these changes are made, you can use the Tanzu Dev Tools extension or other CLI-based debuggers to debug your workload directly on the cluster.

Note

Currently, Developer Conventions only supports debug operations for Java applications.

Next steps

- Install Developer Conventions

Developer Conventions overview

Developer Conventions is a set of conventions that enable your workloads to support live-update and debug operations in Tanzu Application Platform (commonly known as TAP).

Prerequisites

- Tanzu CLI Apps plug-in
- Tanzu Dev Tools for VSCode IDE extension.

Features

Enabling Live Updates

Developer Conventions modifies your workload to enable live updates in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--live-update=true`. For more information about how to deploy a workload with the CLI, see Tanzu apps workload apply.
- You deploy a workload by using the `Tanzu: Live Update Start` option through the Tanzu Developer Tools for VS Code extension. For more information about live updating with the extension, see Overview of Tanzu Developer Tools for Visual Studio Code.

When either of the preceding actions take place, the convention behaves as follows:

1. Looks for the `apps.tanzu.vmware.com/live-update=true` annotation on a PodTemplateSpec associated with a workload.
2. Verifies that the image to which conventions are applied contains a process that can be live updated.
3. Adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod is not scaled down to 0 during a live update session.

After these changes are made, you can use the Tanzu Dev Tools extension or the Tilt CLI to make live update changes to source code directly on the cluster.

Enabling debugging

Developer Conventions modifies your workload to enable debugging in either of the following situations:

- You deploy a workload by using the Tanzu CLI Apps plug-in and include the flag `--debug=true`. For more information about how to deploy a workload with the CLI, see Tanzu apps workload apply.
- You deploy a workload by using the `Tanzu Java Debug Start` option through the Tanzu Developer Tools for VS Code extension. For more information about debugging with the extension, see Overview of Tanzu Developer Tools for Visual Studio Code.

When either of the preceding actions take place, the convention behaves as follows:

1. It looks for the `apps.tanzu.vmware.com/debug=true` annotation on a PodTemplateSpec associated with a workload.
2. It checks for the `debug-8` or `debug-9` labels on the image configuration's bill of materials (BOM).
3. It sets the TimeoutSeconds of the Liveness, Readiness, and Startup probes to 600 if currently set to a lower number.
4. It adds annotations to the PodTemplateSpec to modify the Knative properties `minScale` & `maxScale` such that the minimum and maximum number of pods is 1. This ensures the eventual running pod won't be scaled down to 0 during a debug session.

After these changes are made, you can use the Tanzu Dev Tools extension or other CLI-based debuggers to debug your workload directly on the cluster.

Note

Currently, Developer Conventions only supports debug operations for Java applications.

Next steps

- Install Developer Conventions

Install Developer Conventions

This document tells you how to install Developer Conventions from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Developer Conventions. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before installing Developer Conventions:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Supply Chain Choreographer.

Install

To install Developer Conventions:

1. Get the exact name and version information for the Developer Conventions package to be installed by running:

   ```
   tanzu package available list developer-conventions.tanzu.vmware.com --namespace
   tap-install
   ```

   For example:

```
$ tanzu package available list developer-conventions.tanzu.vmware.com --namespa
ce tap-install
- Retrieving package versions for developer-conventions.tanzu.vmware.com
  NAME                                     VERSION        RELEASED-AT
  developer-conventions.tanzu.vmware.com  0.3.0          2021-10-19T00:00:00Z
```

2. Install the package by running:

```
tanzu package install developer-conventions \
  --package developer-conventions.tanzu.vmware.com \
  --version 0.3.0 \
  --namespace tap-install
```

3. Verify the package install by running:

```
tanzu package installed get developer-conventions --namespace tap-install
```

For example:

```
tanzu package installed get developer-conventions -n tap-install
| Retrieving installation details for developer-conventions...
NAME:                    developer-conventions
PACKAGE-NAME:            developer-conventions.tanzu.vmware.com
PACKAGE-VERSION:         0.3.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

Resource limits

The following resource limits are set on the Developer Conventions service:

```
resources:
  limits:
  cpu: 100m
  memory: 256Mi
  requests:
  cpu: 100m
  memory: 20Mi
```

Uninstall

To uninstall Developer Conventions, follow the guide for Uninstall Tanzu Application Platform packages. The package name for developer conventions is `developer-conventions`.

Run Developer Conventions on an OpenShift cluster

This topic tells you about considerations for running Developer Conventions on OpenShift.

On OpenShift clusters, Developer Conventions must run with a custom SecurityContextConstraint (SCC) to enable compliance with restricted Kubernetes pod security standards. Tanzu Application Platform configures the following SCC for the Developer Convention's webhook when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

Specification follows:

```
---
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: developer-conventions-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - secret
seccompProfiles: []
groups:
  - system:serviceaccounts:developer-conventions
```

Eventing Overview

Eventing in Tanzu Application Platform (commonly known as TAP) is a collection of APIs based on Knative Eventing that allows the use of an event-driven architecture with your applications.

Eventing Overview

Eventing in Tanzu Application Platform (commonly known as TAP) is a collection of APIs based on Knative Eventing that allows the use of an event-driven architecture with your applications.

Install Eventing

This topic tells you how to install the Eventing package from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Eventing. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Eventing:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install Eventing:

1. List version information for the package by running:

   ```
   tanzu package available list eventing.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list eventing.tanzu.vmware.com --namespace tap-instal
   l
   - Retrieving package versions for eventing.tanzu.vmware.com...
     NAME                    VERSION  RELEASED-AT
     eventing.tanzu.vmware.com  2.0.1    2022-10-11T00:00:00Z
   ```

2. (Optional) Make changes to the default installation settings:

   1. Gather values schema.

      ```
      tanzu package available get eventing.tanzu.vmware.com/2.0.1 --values-sche
      ma -n tap-install
      ```

      For example:

      ```
      $ tanzu package available get eventing.tanzu.vmware.com/2.0.1 --values-sc
      hema -n tap-install
      | Retrieving package details for eventing.tanzu.vmware.com/2.0.1...
        KEY           DEFAULT  TYPE     DESCRIPTION
        lite.enable   false    boolean  Optional: Not recommended for productio
      n. Set to "true" to reduce CPU and Memory resource requests for all Event
      ing Deployments, Daemonsets, and Statefulsets by half. On by default when
      "provider" is set to "local".
        pdb.enable    true     boolean  Optional: Set to true to enable Pod Dis
      ruption Budget. If provider local is set to "local", the PDB will be disa
      bled automatically.
        provider      <nil>    string   Optional: Kubernetes cluster provider.
      To be specified if deploying Eventing on a local Kubernetes cluster provi
      der.
      ```

   2. Create a `eventing-values.yaml` by using the following sample `eventing-values.yaml` as a guide:

      ```
      ---
      lite:
        enable: true
      ```

      Note

      For most installations, you can leave the `eventing-values.yaml` empty, and use the default values.

      If you run on a single-node cluster, such as kind or minikube, set the `lite.enable:` property to `true`. This option reduces resources requests for Eventing deployments.

3. Install the package by running:

   ```
   tanzu package install eventing -p eventing.tanzu.vmware.com -v 2.0.1 -n tap-ins
   tall -f eventing-values.yaml --poll-timeout 30m
   ```

For example:

```
$ tanzu package install eventing -p eventing.tanzu.vmware.com -v 2.0.1 -n tap-i
nstall -f eventing-values.yaml --poll-timeout 30m
- Installing package 'eventing.tanzu.vmware.com'
| Getting package metadata for 'eventing.tanzu.vmware.com'
| Creating service account 'eventing-tap-install-sa'
| Creating cluster admin role 'eventing-tap-install-cluster-role'
| Creating cluster role binding 'eventing-tap-install-cluster-rolebinding'
| Creating secret 'eventing-tap-install-values'
| Creating package resource
| Waiting for 'PackageInstall' reconciliation for 'eventing'
| 'PackageInstall' resource install status: Reconciling


Added installed package 'eventing'
```

Use an empty file for `eventing-values.yaml` to enable default installation configuration. Otherwise, see the previous step to set installation configuration values.

4. Verify the package install by running:

```
tanzu package installed get eventing -n tap-install
```

For example:

```
tanzu package installed get eventing -n tap-install
| Retrieving installation details for eventing...
NAME:                    eventing
PACKAGE-NAME:            eventing.tanzu.vmware.com
PACKAGE-VERSION:         2.0.1
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

Overview of Flux CD Source Controller

Flux CD Source Controller provides you with APIs for acquiring resources such as Git, Helm repositories and S3 buckets on the cluster. For more information, see Flux CD Source Controller documentation.

The source-controller implements the source.toolkit.fluxcd.io API in GitHub and is a core component of the GitOps toolkit.

Overview of Flux CD Source Controller

Flux CD Source Controller provides you with APIs for acquiring resources such as Git, Helm repositories and S3 buckets on the cluster. For more information, see Flux CD Source Controller documentation.

The source-controller implements the source.toolkit.fluxcd.io API in GitHub and is a core component of the GitOps toolkit.

Install Flux CD Source Controller

This topic tells you how to install Flux CD Source Controller from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Flux CD Source Controller. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Flux CD Source Controller:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager on the cluster. For more information, see Install cert-manager.

Configuration

The Flux CD Source Controller package has no configuration values.

Installation

To install Flux CD Source Controller from the Tanzu Application Platform package repository:

1. List version information for the package by running:

   ```
   tanzu package available list fluxcd.source.controller.tanzu.vmware.com -n tap-install
   ```

   For example:

   ```
   $ tanzu package available list fluxcd.source.controller.tanzu.vmware.com -n tap-install
       \ Retrieving package versions for fluxcd.source.controller.tanzu.vmware.com...
         NAME                                        VERSION  RELEASED-AT
         fluxcd.source.controller.tanzu.vmware.com   0.16.0   2021-10-27 19:00:00 -0500 -05
   ```

2. Install the package by running:

   ```
   tanzu package install fluxcd-source-controller -p fluxcd.source.controller.tanzu.vmware.com -v VERSION-NUMBER -n tap-install
   ```

   Where:

   - `VERSION-NUMBER` is the version of the package listed in step 1.

   For example:

   ```
   tanzu package install fluxcd-source-controller -p fluxcd.source.controller.tanzu.vmware.com -v 0.16.0 -n tap-install
   \ Installing package 'fluxcd.source.controller.tanzu.vmware.com'
   | Getting package metadata for 'fluxcd.source.controller.tanzu.vmware.com'
   | Creating service account 'fluxcd-source-controller-tap-install-sa'
   | Creating cluster admin role 'fluxcd-source-controller-tap-install-cluster-role'
   | Creating cluster role binding 'fluxcd-source-controller-tap-install-cluster-rolebinding'
   | Creating package resource
   - Waiting for 'PackageInstall' reconciliation for 'fluxcd-source-controller'
   ```

```
| 'PackageInstall' resource install status: Reconciling

  Added installed package 'fluxcd-source-controller'
```

This package creates a new namespace called `flux-system`. This namespace hosts all the elements of fluxcd.

3. Verify the package install by running:

```
tanzu package installed get fluxcd-source-controller -n tap-install
```

For example:

```
tanzu package installed get fluxcd-source-controller -n tap-install
\ Retrieving installation details for fluxcd-source-controller...
NAME:                    fluxcd-source-controller
PACKAGE-NAME:            fluxcd.source.controller.tanzu.vmware.com
PACKAGE-VERSION:         0.16.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

```
kubectl get pods -n flux-system
```

For example:

```
$ kubectl get pods -n flux-system
NAME                                 READY   STATUS    RESTARTS   AGE
source-controller-69859f545d-ll8fj   1/1     Running   0          3m38s
```

Verify that `STATUS` is `Running`.

Try fluxcd-source-controller

1. Verify the main components of `fluxcd-source-controller` were installed by running:

```
kubectl get all -n flux-system
```

Expect to see the following outputs or similar:

```
NAME                                     READY   STATUS    RESTARTS   AGE
pod/source-controller-7684c85659-2zfxb   1/1     Running   0          40m

NAME                        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/source-controller   ClusterIP   10.108.138.74   <none>        80/TCP
40m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/source-controller   1/1     1            1           40m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/source-controller-7684c85659   1         1         1       40m
```

2. Verify all the CRD were installedby running:

```
kubectl get crds -n flux-system | grep ".fluxcd.io"
buckets.source.toolkit.fluxcd.io                          2022-03-07T19:20:14Z
gitrepositories.source.toolkit.fluxcd.io                  2022-03-07T19:20:14Z
```

```
helmcharts.source.toolkit.fluxcd.io                    2022-03-07T19:20:14Z
helmrepositories.source.toolkit.fluxcd.io              2022-03-07T19:20:14Z
```

Note

You will communicate with `fluxcd-source-controller` through its CRDs.

3. Follow these steps to consume a `GitRepository` object:
    1. Create the following `gitrepository-sample.yaml` file:

        ```
        apiVersion: source.toolkit.fluxcd.io/v1beta1
        kind: GitRepository
        metadata:
          name: gitrepository-sample
        spec:
          interval: 1m
          url: https://github.com/vmware-tanzu/application-accelerator-samples
          ref:
            branch: main
        ```

    2. Apply the created conf:

        ```
        kubectl apply -f gitrepository-sample.yaml
        gitrepository.source.toolkit.fluxcd.io/gitrepository-sample created
        ```

    3. Verify the git-repository was fetched correctly:

        ```
        kubectl get GitRepository
        NAME                    URL
        READY   STATUS
        AGE
        gitrepository-sample    https://github.com/vmware-tanzu/application-accele
        rator-samples    True    Fetched revision: main/132f4e719209eb10b9485302f8
        593fc0e680f4fc   4s
        ```

    For more examples, see the samples directory on fluxcd/source-controller/samples in GitHub.

Documentation

For documentation specific to fluxcd-source-controller, see the main repository fluxcd/source-controller in GitHub.

Overview of Learning Center for Tanzu Application Platform

Learning Center provides a platform for creating and self-hosting workshops. It allows you to create workshops from markdown files that are displayed to the learner in a terminal shell environment with an instructional wizard UI.

The UI can embed slide content, an integrated development environment (IDE), a web console for accessing the Kubernetes cluster, and other custom web applications.

Although Learning Center requires Kubernetes to run, and is used to teach users about Kubernetes, you can use it to host training for other purposes as well. For example, you can use it to help train users in web-based applications, use of databases, or programming languages, where the user has no interest or need for Kubernetes.

Use cases

Use case scenarios that Learning Center supports include:

- Supervised workshops. For example, a workshop run at a conference, at a customer site, or online. The workshop has a set time period and you know the maximum number of users to expect. After the training is complete, the Kubernetes cluster created for the workshop is destroyed.
- Temporary learning portal. This is for when you must provide access to a small set of workshops for a short duration for hands on demos at a conference vendor booth. Users select which topic they want to learn about and do that workshop. The workshop instance is created on demand. When they have finished the workshop, that workshop instance is destroyed to free up resources. After the conference has finished, the Kubernetes cluster is destroyed.
- Permanent learning portal. Similar to the temporary learning portal, but runs on an extended basis as a public website where anyone can come and learn at any time.
- Personal training or demos. This is where anyone who wants to run a workshop on their own Kubernetes cluster to learn that topic, or where a product demo was packaged up as a workshop and they want to use it to demonstrate the product to a customer. You can destroy the workshop environment when complete, but there is no need for the cluster to be destroyed.

When running workshops, wherever possible a shared Kubernetes cluster reduces the amount of setup required. This works for developer-focused workshops, becauses it is usually not necessary to provide elevated access to the Kubernetes cluster, and you can use role-based access controls (RBAC) to prevent users from interfering with each other. You can also set quotas so users are restricted as to how much resources they can use.

When you run workshops that deal with cluster operations, for which users need cluster admin access, create a separate cluster for each user. Learning Center doesn't deal with provisioning clusters, only with deploying a workshop environment in a cluster after it exists.

Use case requirements

In implementing to the preceding scenarios, the primary requirements related to creation of workshop content, and what you can do at runtime, are as follows:

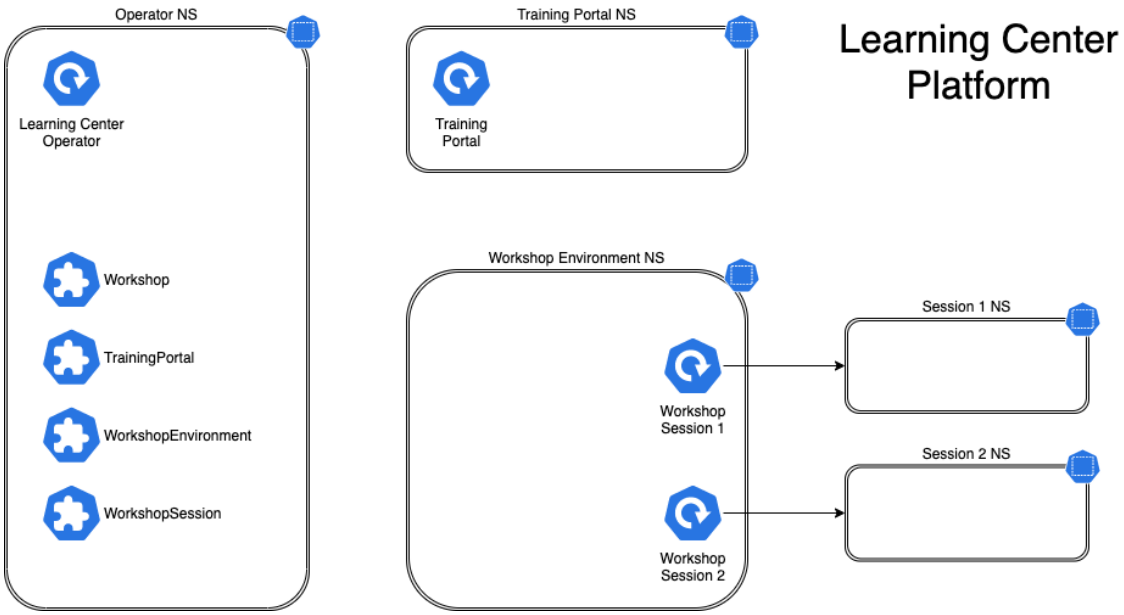
- You must store everything for the workshop in a Git repository, with no dependency on using a special web application or service to create a workshop.
- Use GitHub as a means to distribute workshop content. Alternatively, you can distribute the workshop as a container image. The latter is necessary if special tools must be installed for use in a workshop.
- Provide instructions to the user to complete the workshop as Markdown or AsciiDoc files.
- You can annotate instructions as executable commands so that when clicked in the workshop dashboard, they execute for the user in the appropriate terminal to avoid mistakes when commands are entered manually.
- You can annotate text as copyable so when clicked in the workshop dashboard, it is copied into the browser paste buffer ready for pasting into the terminal or other web application.
- Provide each user access to one or more namespaces in the Kubernetes cluster unique to their session. For Kubernetes based workshops, this is where applications are deployed as part of the workshop.

- You can create additional Kubernetes resources specific to a workshop session in advance of the session. This enables the deployment of applications for each user session.
- You can deploy additional Kubernetes resources common to all workshop sessions when the workshop environment is first created. This enables deployment of applications shared by all users.
- Apply resource quotas on each workshop session to control how much resources users can consume.
- Apply role-based access control (RBAC) on each workshop session to control what users can do.
- Provide access to an editor (IDE) in the workshop dashboard in the web browser for users to edit files during the workshop.
- Provide access to a web-based console for accessing the Kubernetes cluster. Use of the Kubernetes dashboard or Octant is supported.
- Ability to integrate additional web-based applications into the workshop dashboard specific to the topic of the workshop.
- Ability for the workshop dashboard to display slides used by an instructor in support of the workshop.

Platform architectural overview

The Learning Center relies on a Kubernetes Operator to perform the bulk of the work. The actions of the operator are controlled by using a set of custom resources specific to the Learning Center.

There are multiple ways of using the custom resources to deploy workshops. The primary way is to create a training portal, which in turn then triggers the setup of one or more workshop environments, one for each distinct workshop. When users access the training portal and select the workshop they want to do, the training portal allocates to that user a workshop session (creating one if necessary) against the appropriate workshop environment, and the user is redirected to that workshop session instance.



You can associate each workshop session with one or more Kubernetes namespaces specifically for use during that session. Role-Based Access Control (RBAC) applied to the unique Kubernetes service account for that session ensures that the user can only access the namespaces and other resources that they are allowed to for that workshop.

In this scenario, the custom resource types that come into play are:

- `Workshop` - Provides the definition of a workshop. Preloaded by an admin into the cluster, it defines where the workshop content is hosted, or the location of a container image which bundles the workshop content and any additional tools required for the workshop. The definition also lists additional resources that must be created which are to be shared between all workshop sessions, or for each session, with details of resources quotas and access roles required by the workshop.
- `TrainingPortal` - Created by an admin in the cluster to trigger the deployment of a training portal. The training portal can provide access to one or more distinct workshops defined by a `Workshop` resource. The training portal provides a web based interface for registering for workshops and accessing them. It also provides a REST API for requesting access to workshops, allowing custom front ends to be created which integrate with separate identity providers and which provide an alternate means for browsing and accessing workshops.
- `WorkshopEnvironment` - Used by the training portal to trigger the creation of a workshop environment for a workshop. This causes the operator to set up a namespace for the workshop into which shared resources are deployed, and where the workshop sessions are run.
- `WorkshopSession` - Used by the training portal to trigger the creation of a workshop session against a specific workshop environment. This causes the operator to set up any namespaces specific to the workshop session and pre-create additional resources required for a workshop session. Workshop sessions can either be created up front in reserve, to be handed out when requested, or created on demand.

Next steps

Learn more about:

- Workshops
- Get started with Learning Center
- Installing Learning Center
- Local install guides
- Air-gapped environment requirements

Overview of Learning Center for Tanzu Application Platform

Learning Center provides a platform for creating and self-hosting workshops. It allows you to create workshops from markdown files that are displayed to the learner in a terminal shell environment with an instructional wizard UI.

The UI can embed slide content, an integrated development environment (IDE), a web console for accessing the Kubernetes cluster, and other custom web applications.

Although Learning Center requires Kubernetes to run, and is used to teach users about Kubernetes, you can use it to host training for other purposes as well. For example, you can use it to help train users in web-based applications, use of databases, or programming languages, where the user has no interest or need for Kubernetes.

Use cases

Use case scenarios that Learning Center supports include:

- Supervised workshops. For example, a workshop run at a conference, at a customer site, or online. The workshop has a set time period and you know the maximum number of users to expect. After the training is complete, the Kubernetes cluster created for the workshop is destroyed.
- Temporary learning portal. This is for when you must provide access to a small set of workshops for a short duration for hands on demos at a conference vendor booth. Users select which topic they want to learn about and do that workshop. The workshop instance is created on demand. When they have finished the workshop, that workshop instance is destroyed to free up resources. After the conference has finished, the Kubernetes cluster is destroyed.
- Permanent learning portal. Similar to the temporary learning portal, but runs on an extended basis as a public website where anyone can come and learn at any time.
- Personal training or demos. This is where anyone who wants to run a workshop on their own Kubernetes cluster to learn that topic, or where a product demo was packaged up as a workshop and they want to use it to demonstrate the product to a customer. You can destroy the workshop environment when complete, but there is no need for the cluster to be destroyed.

When running workshops, wherever possible a shared Kubernetes cluster reduces the amount of setup required. This works for developer-focused workshops, becauses it is usually not necessary to provide elevated access to the Kubernetes cluster, and you can use role-based access controls (RBAC) to prevent users from interfering with each other. You can also set quotas so users are restricted as to how much resources they can use.

When you run workshops that deal with cluster operations, for which users need cluster admin access, create a separate cluster for each user. Learning Center doesn't deal with provisioning clusters, only with deploying a workshop environment in a cluster after it exists.

Use case requirements

In implementing to the preceding scenarios, the primary requirements related to creation of workshop content, and what you can do at runtime, are as follows:

- You must store everything for the workshop in a Git repository, with no dependency on using a special web application or service to create a workshop.
- Use GitHub as a means to distribute workshop content. Alternatively, you can distribute the workshop as a container image. The latter is necessary if special tools must be installed for use in a workshop.
- Provide instructions to the user to complete the workshop as Markdown or AsciiDoc files.
- You can annotate instructions as executable commands so that when clicked in the workshop dashboard, they execute for the user in the appropriate terminal to avoid mistakes when commands are entered manually.
- You can annotate text as copyable so when clicked in the workshop dashboard, it is copied into the browser paste buffer ready for pasting into the terminal or other web application.
- Provide each user access to one or more namespaces in the Kubernetes cluster unique to their session. For Kubernetes based workshops, this is where applications are deployed as part of the workshop.
- You can create additional Kubernetes resources specific to a workshop session in advance of the session. This enables the deployment of applications for each user session.
- You can deploy additional Kubernetes resources common to all workshop sessions when the workshop environment is first created. This enables deployment of applications shared

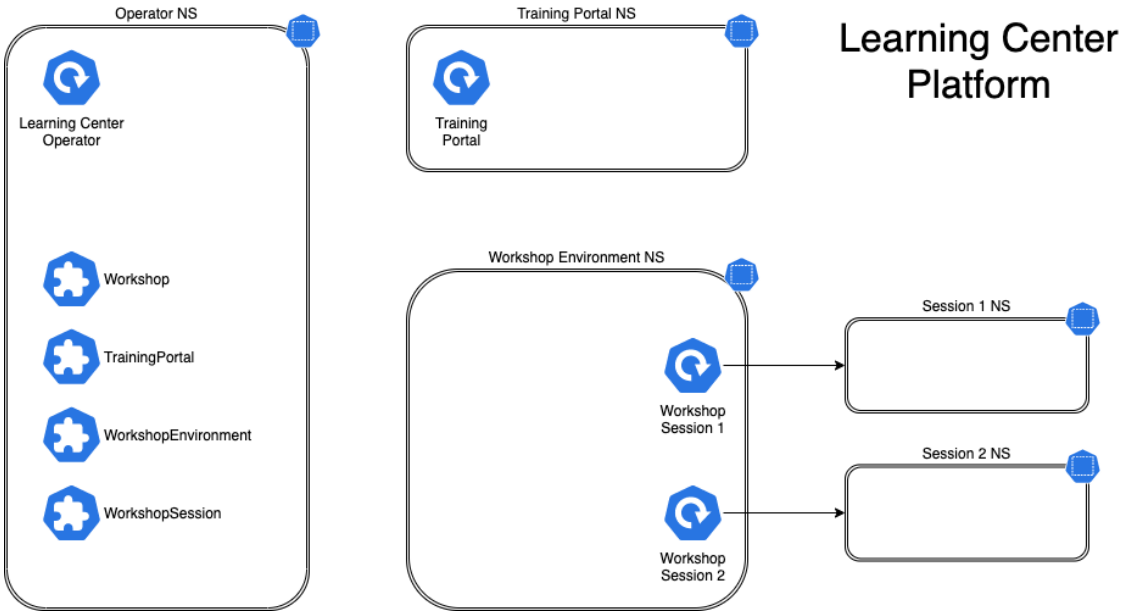
by all users.

- Apply resource quotas on each workshop session to control how much resources users can consume.
- Apply role-based access control (RBAC) on each workshop session to control what users can do.
- Provide access to an editor (IDE) in the workshop dashboard in the web browser for users to edit files during the workshop.
- Provide access to a web-based console for accessing the Kubernetes cluster. Use of the Kubernetes dashboard or Octant is supported.
- Ability to integrate additional web-based applications into the workshop dashboard specific to the topic of the workshop.
- Ability for the workshop dashboard to display slides used by an instructor in support of the workshop.

Platform architectural overview

The Learning Center relies on a Kubernetes Operator to perform the bulk of the work. The actions of the operator are controlled by using a set of custom resources specific to the Learning Center.

There are multiple ways of using the custom resources to deploy workshops. The primary way is to create a training portal, which in turn then triggers the setup of one or more workshop environments, one for each distinct workshop. When users access the training portal and select the workshop they want to do, the training portal allocates to that user a workshop session (creating one if necessary) against the appropriate workshop environment, and the user is redirected to that workshop session instance.



You can associate each workshop session with one or more Kubernetes namespaces specifically for use during that session. Role-Based Access Control (RBAC) applied to the unique Kubernetes service account for that session ensures that the user can only access the namespaces and other resources that they are allowed to for that workshop.

In this scenario, the custom resource types that come into play are:

- `Workshop` - Provides the definition of a workshop. Preloaded by an admin into the cluster, it defines where the workshop content is hosted, or the location of a container image which

bundles the workshop content and any additional tools required for the workshop. The definition also lists additional resources that must be created which are to be shared between all workshop sessions, or for each session, with details of resources quotas and access roles required by the workshop.

- `TrainingPortal` - Created by an admin in the cluster to trigger the deployment of a training portal. The training portal can provide access to one or more distinct workshops defined by a `Workshop` resource. The training portal provides a web based interface for registering for workshops and accessing them. It also provides a REST API for requesting access to workshops, allowing custom front ends to be created which integrate with separate identity providers and which provide an alternate means for browsing and accessing workshops.
- `WorkshopEnvironment` - Used by the training portal to trigger the creation of a workshop environment for a workshop. This causes the operator to set up a namespace for the workshop into which shared resources are deployed, and where the workshop sessions are run.
- `WorkshopSession` - Used by the training portal to trigger the creation of a workshop session against a specific workshop environment. This causes the operator to set up any namespaces specific to the workshop session and pre-create additional resources required for a workshop session. Workshop sessions can either be created up front in reserve, to be handed out when requested, or created on demand.

Next steps

Learn more about:

- Workshops
- Get started with Learning Center
- Installing Learning Center
- Local install guides
- Air-gapped environment requirements

Install Learning Center

This topic describes how to install Learning Center from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Learning Center. For more information about profiles, see Components and installation profiles.

To install Tanzu Learning Center, see the following sections.

For general information about Learning Center, see Learning Center. For information about deploying Learning Center operator, see Install and configure the Learning Center operator.

Prerequisites

Before installing Learning Center:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- The cluster must have an ingress router configured. If you have installed Learning Center through a profile, it already deploys a Contour ingress controller.
- The operator, when deploying instances of the workshop environments, must be able to expose them through an external URL for access. For the custom domain you are using, DNS must have been configured with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.
- By default, the workshop portal and workshop sessions are accessible over HTTP connections. If you wish to use secure HTTPS connections, you must have access to a wildcard SSL certificate for the domain under which you want to host the workshops. You cannot use a self-signed certificate.
- Any ingress routes created use the default ingress class if you have multiple ingress class types available and you must override which is used.

Install Learning Center

To install Learning Center:

1. List version information for the package by running:

   ```
   tanzu package available list learningcenter.tanzu.vmware.com --namespace tap-in
   stall
   ```

   Example output:

   ```
   NAME                              VERSION        RELEASED-AT
   learningcenter.tanzu.vmware.com  0.1.0          2021-12-01 08:18:48 -0500 EDT
   ```

2. (Optional) See all the configurable parameters on this package by running:

   Remember to change the 0.x.x version

   ```
   tanzu package available get learningcenter.tanzu.vmware.com/0.x.x --values-sche
   ma --namespace tap-install
   ```

3. Create a config file named `learning-center-config.yaml`.

4. To override the `shared.ingress_domain` in the values file of Tanzu Application Platform, add the parameter `ingressDomain` to `learning-center-config.yaml`. For example:

   ```
   ingressDomain: YOUR-INGRESS-DOMAIN
   ```

   Where `YOUR-INGRESS-DOMAIN` is the domain name for your Kubernetes cluster.

   When deploying workshop environment instances, the operator must be able to expose the instances through an external URL. You need this access to discover the domain name that can be used as a suffix to host names for instances.

   For the custom domain you are using, DNS must have been configured with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.

   If you are running Kubernetes on your local machine using a system such as `minikube` and you don't have a custom domain name that maps to the IP address for the cluster, you can use a `nip.io` address. For example, if `minikube ip` returns `192.168.64.1`, you can use the `192.168.64.1.nip.io` domain. You cannot use an address of form `127.0.0.1.nip.io` or

`subdomain.localhost`. This causes a failure. Internal services needing to connect to each other connect to themselves instead, because the address resolves to the host loopback address of `127.0.0.1`.

5. Add the `ingressSecret` to `learning-center-config.yaml`, as in this example:

```
ingressSecret:
certificate: |
  -----BEGIN CERTIFICATE-----
  MIIFLTCCBBWgAwIBAgaSAys/V2NCTG9uXa9aAiYt7WJ3MA0GCSqGaIb3DQEBCwUA
                                    ...
  dHa6Ly9yMy5vamxlbmNyLm9yZzAiBggrBgEFBQawAoYWaHR0cDoaL3IzLmkubGVu
  -----END CERTIFICATE-----
privateKey: |
  -----BEGIN PRIVATE KEY-----
  MIIEvQIBADAaBgkqhkiG9waBAQEFAASCBKcwggSjAgEAAoIBAaCx4nyc2xwaVOzf
                                    ...
  IY/9SatMcJZivH3F1a7SXL98PawPIOSR7986P7rLFHzNjaQQ0DWTaXBRt+oUDxpN
  -----END PRIVATE KEY-----
```

If you already have a TLS secret, follow these steps **before deploying any workshop**: - Create the `learningcenter` namespace manually or the one you defined - Copy the TLS secret to the `learningcenter` namespace or the one you defined and use the `secretName` property as in this example:

```
ingressSecret:
 secretName: workshops.example.com-tls
```

By default, the workshop portal and workshop sessions are accessible over HTTP connections.

To use secure HTTPS connections, you must have access to a wildcard SSL certificate for the domain under which you want to host the workshops. You cannot use a self-signed certificate.

You can create wildcard certificates by using letsencrypt https://letsencrypt.org/_. After you have the certificate, you can define the `certificate` and `privateKey` properties under the `ingressSecret` property to specify the certificate on the configuration YAML.

6. Any ingress routes created use the default ingress class. If you have multiple ingress class types available, and you need to override which is used, define the `ingressClass` property in `learning-center-config.yaml` **before deploying any workshop**:

```
ingressClass: contour
```

7. Install Learning Center operator by running:

   Remember to change the 0.x.x version

```
tanzu package install learning-center --package-name learningcenter.tanzu.vmwar
e.com --version 0.x.x -f learning-center-config.yaml
```

The preceding command creates a default namespace in your Kubernetes cluster called `learningcenter`, and the operator, and any required namespaced resources, are created in it. A set of custom resource definitions and a global cluster role binding are also created.

You can confirm that the operator deployed successfully by running:

```
kubectl get all -n learningcenter
```

The pod for the operator should be marked as running.

Install the Self-Guided Tour Training Portal and Workshop

To install the Self-Guided Tour Training Portal and Workshop:

1. Confirm you have the workshop package installed by running:

   ```
   tanzu package available list workshops.learningcenter.tanzu.vmware.com --namesp
   ace tap-install
   ```

2. Install the Learning Center Training Portal with the Self-Guided Tour Workshop by running:

   Remember to change the 0.x.x version

   ```
   tanzu package install learning-center-workshop --package-name workshops.learnin
   gcenter.tanzu.vmware.com --version 0.x.x -n tap-install
   ```

3. Check for the Training Portals available in your environment by running:

   ```
   kubectl get trainingportals
   ```

   Example output:

   ```
   NAME                        URL                                                ADMINU
   SERNAME         ADMINPASSWORD                       STATUS
       learningcenter-tutorials   http://learningcenter-tutorials.example.com   le
   arningcenter        QGBaM4CF01toPiZLW5NrXTcIYSpw2UJK   Running
   ```

Supported Learning Center Values Configuration

Admins are provided the following sample learning-center-config.yaml file to see the possible configurations supported by Learning Center. These configurations are additional ones that admins can provide to the operator resource but are by no means necessary for Learning Center to work. It is enough to follow the previous instructions on this page for Learning Center to run.

It is important to note that Learning Center has default values in place for the learning-center-config.yaml file. Admins only need to provide the values they want to override. As in the example above, overriding the ingressDomain property is enough to get Learning Center to work.

```
#! The namespace in which to deploy Learning Center. For now this must be "learningcen
ter" as
namespace: learningcenter
#! DNS parent subdomain used for training portal and workshop ingresses.
ingressDomain: workshops.example.com
#! Ingress class for where multiple ingress controllers exist and need to
#! use that which is not marked as the default.
ingressClass: null
#! SSL certificate for secure ingress. This must be a wildcard certificate for
#! children of DNS parent ingress subdomain.
ingressSecret:
  certificate: null
  privateKey: null
  secretName: null
#! Configuration for persistent volumes. The default storage class specified
#! by the cluster is used if not defined. You might need to set storage group
#! where a cluster has pod security policies enabled, usually
#! to one. Set storage user and storage group in exceptional cases
#! where storage class uses maps to NFS storage and storage server requires
#! that a specific user and group always be used.
storageClass: null
storageUser: null
storageGroup: null
```

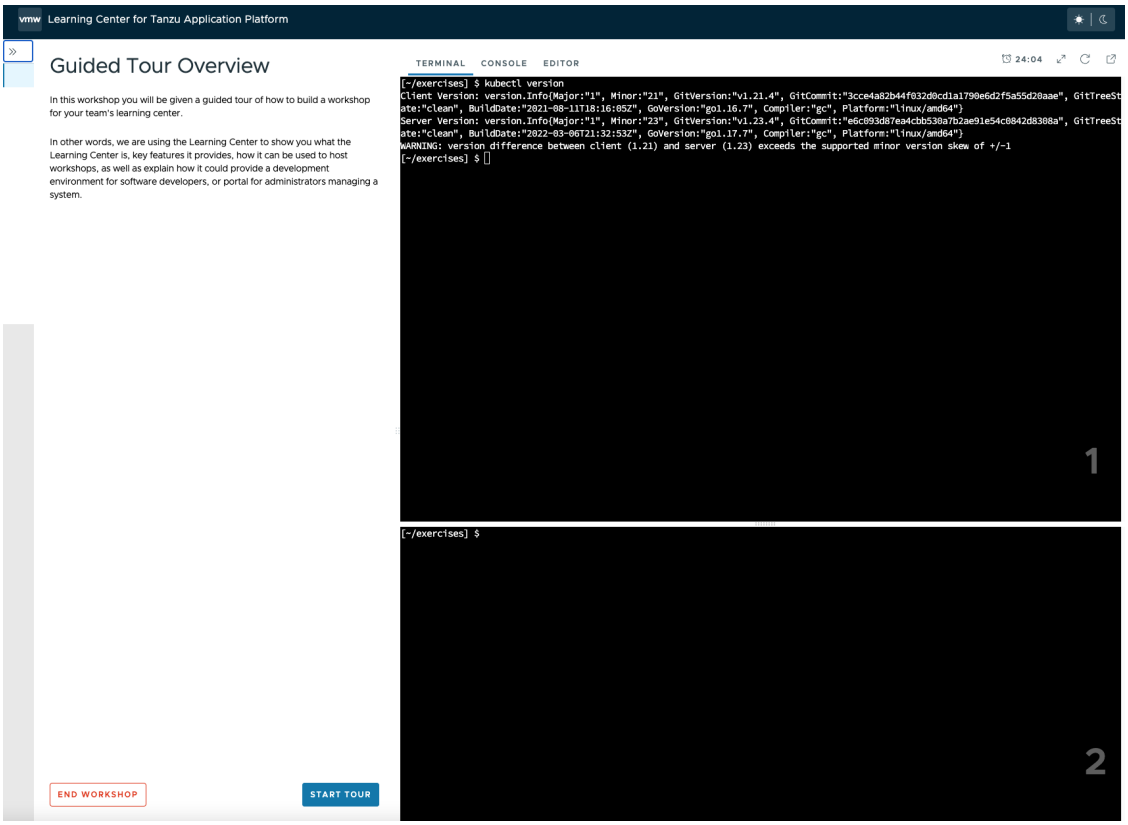
```
#! Credentials for accessing training portal instances. If not specified,
#! random passwords are generated that you can obtain from the custom resource
#! for the training portal.
portalCredentials:
  systemAdmin:
    username: learningcenter
    password: null
  clientAccess:
    username: robot@learningcenter
    password: null
#! Container image versions for various components of Learning Center. The Learning Ce
nter
#! operator needs to be modified to read names of images for the registry
#! and docker-in-docker from config map to enable disconnected install.
#! Prepull images to nodes in cluster. Should be an empty list if no images
#! should be prepulled. Normally you would only want to prepull workshop
#! images. This is done to reduce start-up times for sessions.
prepullImages: ["base-environment"]
#! Docker daemon settings when building docker images in a workshop is
#! enabled. Proxy cache provides a way of partially getting around image
#! pull limits for Docker Hub image registry, with the remote URL being
#! set to "https://registry-1.docker.io".
dockerDaemon:
  networkMTU: 1500
  proxyCache:
    remoteURL: null
    username: null
    password: null
#! Used to restrict access to IP addresses or IP subnets. This must be a CIDR block ra
nge corresponding to the subnet or a portion of a
#! subnet you want to block. A Kubernetes `NetworkPolicy` is used to enforce the restr
iction. So the
#! Kubernetes cluster must use a network layer supporting network policies, and the ne
cessary Kubernetes
#! controllers supporting network policies must be enabled when the cluster is install
ed.
network:
  blockCIDRs:
  - 169.254.169.254/32
  - fd00:ec2::254/128
```

See Restricting Network Access for more information on blocking CIDRs.

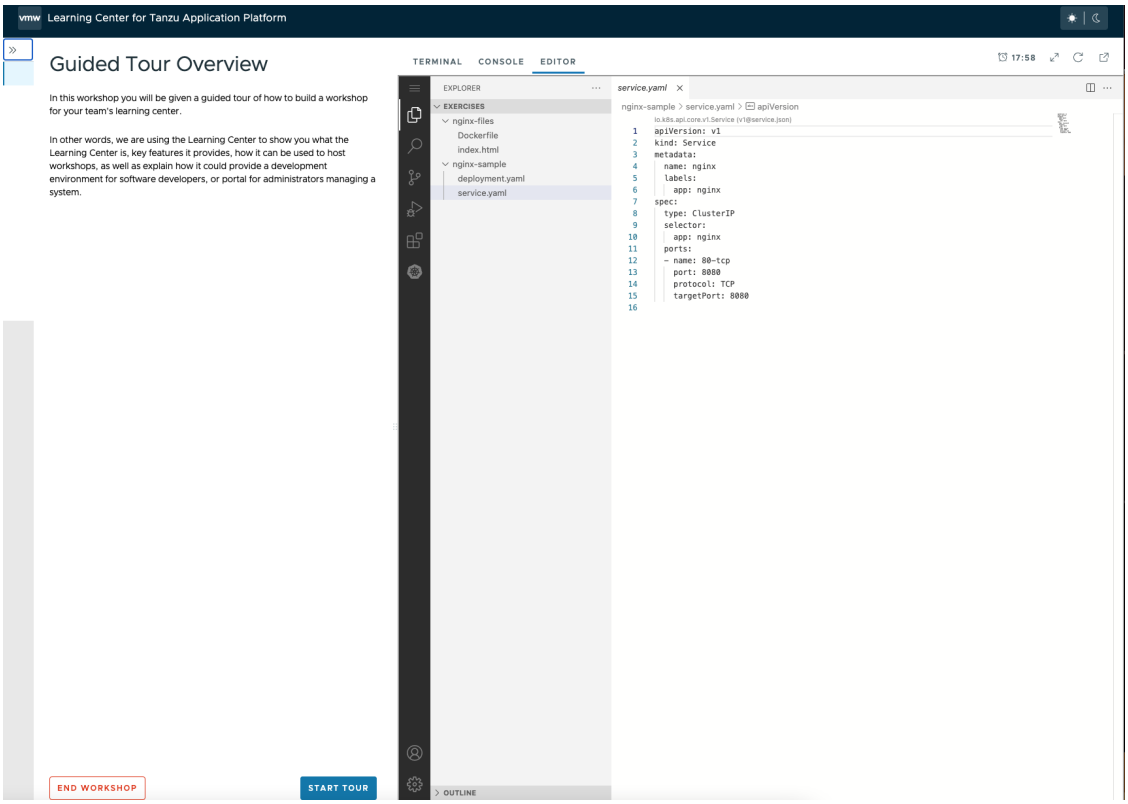
About Learning Center workshops

This topic gives you an overview of Learning Center workshops.

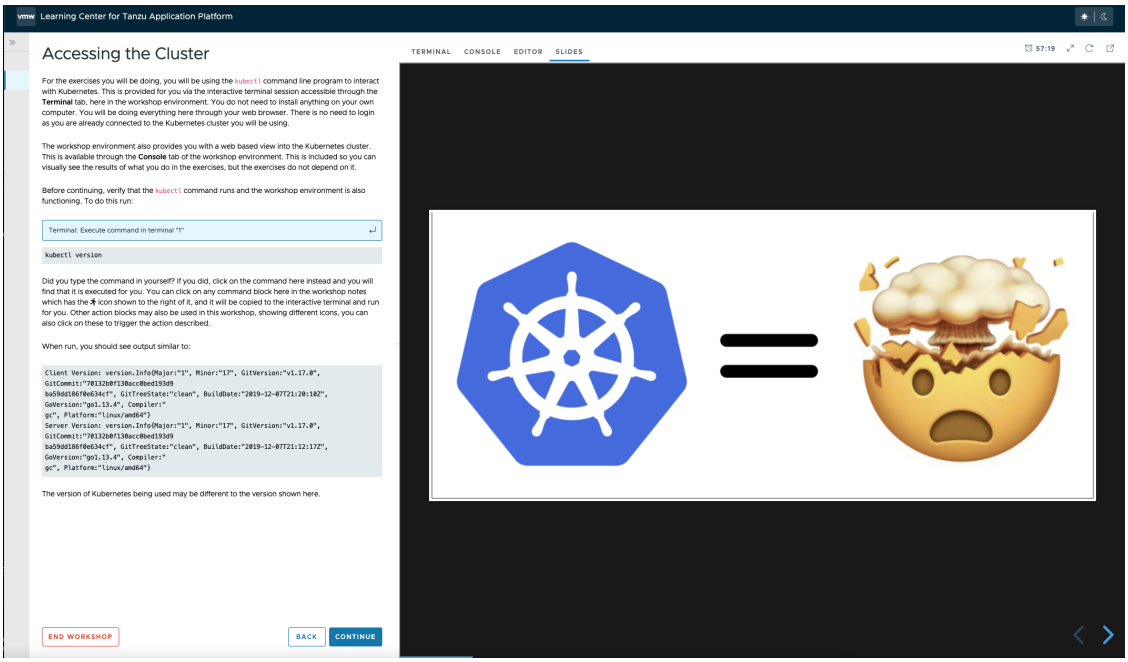
The Learning Center workshop dashboard comprises a set of workshop instructions on the left-hand side and a series of tabbed views on the right-hand side. For workshops that require users to run commands, one or more terminal shells are provided. For more information about workshops including creating your own, see Create workshops.



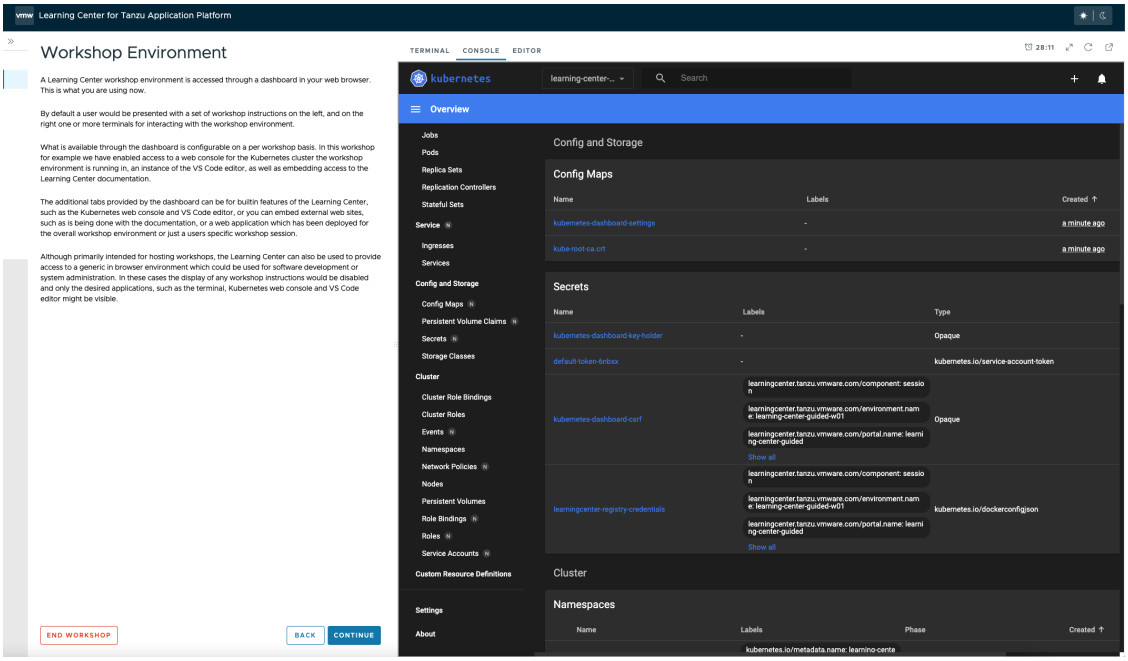
The terminals provide access to the editors `vi` and `nano`. To provide a UI based editor, you can enable the embedded editor view and use the embedded IDE based on VS Code.



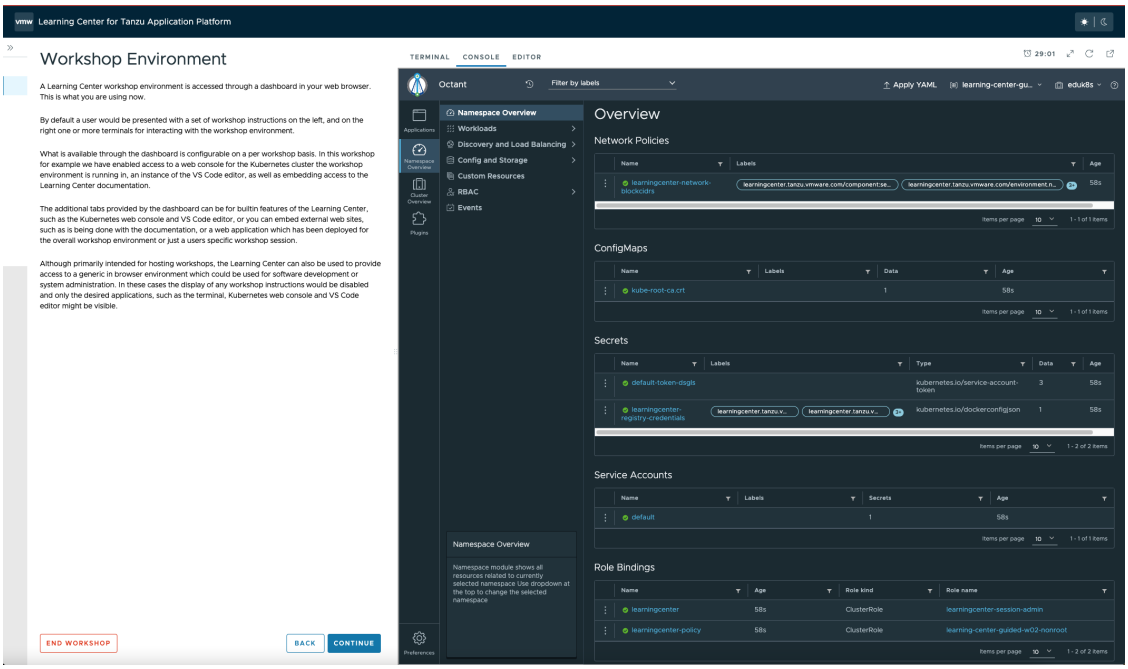
To complement the workshop instructions, or to be available for use by the instructor, you can include slides with a workshop. For slides you can use HTML based slide presentation tools such as `reveal.js`, or you can embed a PDF file.



If the workshop involves working with Kubernetes, you can enable a web console for accessing the Kubernetes cluster. The default web console uses the Kubernetes dashboard.



Alternatively, you can enable Octant as the web console.



Get started with Learning Center

This topic describes how you can get started with Learning Center for Tanzu Application Platform. For information about Learning Center and its use cases, see Learning Center for Tanzu Application Platform.

Installing Learning Center

Before deploying workshops, you must install a Kubernetes operator for Learning Center. The operator manages the setup of the environment for each workshop and deploys instances of a workshop for each person.

For basic information about installing the Learning Center operator, see Install Learning Center.

Get started

See the following useful information about getting started with Learning Center:

- Install and configure the Learning Center operator
- Get started with workshops
- Get started with training portals
- Delete an operator

Get started with Learning Center

This topic describes how you can get started with Learning Center for Tanzu Application Platform. For information about Learning Center and its use cases, see Learning Center for Tanzu Application Platform.

Installing Learning Center

Before deploying workshops, you must install a Kubernetes operator for Learning Center. The operator manages the setup of the environment for each workshop and deploys instances of a

workshop for each person.

For basic information about installing the Learning Center operator, see Install Learning Center.

Get started

See the following useful information about getting started with Learning Center:

- Install and configure the Learning Center operator
- Get started with workshops
- Get started with training portals
- Delete an operator

Install and configure the Learning Center operator

This topic gives you information about installing and configuring the Learning Center operator.

Before deploying workshops, you must install a Kubernetes operator for Learning Center. The operator manages the setup of the environment for each workshop and deploys instances of a workshop for each person.

For basic information about installing the operator, see Install Learning Center.

Installing and setting up Learning Center operator

You can deploy the Learning Center operator to any Kubernetes cluster supporting custom resource definitions and the concept of operators. The cluster must have an ingress router configured, though only a basic deployment of the ingress controller is usually required. You do not need to configure the ingress controller to handle cluster wide edge termination of secure HTTP connections. Learning Center creates Kubernetes Ingress resources and supplies any secret for use with secure HTTP connections for each ingress.

For the ingress controller, VMware recommends the use of Contour over alternatives such as nginx. An nginx-based ingress controller has a less than optimal design. Every time a new ingress is created or deleted, the nginx config is reloaded. This causes websocket connections to terminate after a period of time. Learning Center terminals reconnect automatically in the case of the websocket connection being lost. However, not all applications you might use with specific workshops can handle loss of websocket connections so gracefully, and they might be impacted due to the use of an nginx ingress controller. This problem is not specific to Learning Center. It can impact any application when an nginx ingress controller is used frequently and ingresses are created or deleted frequently.

You can use a hosted Kubernetes solution from an IaaS provider such as Google, AWS, or Azure. If you do, as needed increase any HTTP request timeout specified on the inbound load balancer for the ingress controller so that you can use long-lived websocket connections. In some cases, load balancers of hosted Kubernetes solutions only have a 30-second timeout. If possible, configure the timeout applying to websockets to be 1 hour.

If you deploy the web-based training portal, the cluster must have available persistent volumes of type `ReadWriteOnce (RWO)`. A default storage class must be defined so that persistent volume claims do not need to specify a storage class. For some Kubernetes distributions, including from IBM, you must configure Learning Center as to what user and group must be used for persistent volumes. If no default storage class is specified, or a specified storage class is required, you can configure Learning Center with the name of the storage class.

To install the Learning Center operator, you must have cluster admin access.

Cluster pod security policies

The Learning Center operator defines pod security policies to limit what users can do from workshops when deploying workloads to the cluster. The default policy prohibits running of images as the `root` user or using a privileged pod. Specified workshops can relax these restrictions and apply a policy that enables additional privileges required by the workshop.

To enforce a security policy around what a user can do, different mechanisms have been provided with standard Kubernetes distributions and derivatives such as OpenShift. These are:

- Pod security policies (Kubernetes <= 1.25)
- Pod security standards (Kubernetes >= 1.22)
- Security context constraints (OpenShift)

For pod security policies and pod security standards, these both must be enabled in the Kubernetes cluster at the time the cluster is created. They cannot be enabled afterwards. For some Kubernetes distributions, such as Tanzu Community Edition, it is not possible to enable pod security policies. Because pod security standards are new, they might also not be supported.

VMware recommends that the pod security policy admission controller be enabled for the cluster to ensure that the pod security policies are applied. If the admission controller is not enabled, users can deploy workloads that run as the `root` user in a container, or run privileged pods.

If you are unable to enable the pod security policy admission controller, you should only provide access to workshops deployed using the Learning Center operator to users you trust.

Whether the absence of the pod security policy admission controller causes issues with access to persistent volumes depends on the cluster. Although minikube does not enable the pod security policy admission controller, it works as persistent volumes when mounted to give write permissions to all users.

No matter whether pod security policies are enabled, individual workshops must be reviewed as to what added privileges they grant before allowing their use in a cluster.

Specifying the ingress domain

When deploying instances of workshop environments, the operator must expose the instances by using an external URL for access to define the domain name that is used as a suffix to host names for instances.

Note

For the custom domain you are using, configure your DNS with a wildcard domain to forward all requests for subdomains of the custom domain to the ingress router of the Kubernetes cluster.

VMware recommends that you avoid using a `.dev` or `.app` domain name, because such domain names require browsers to use HTTPS and not HTTP. Although you can provide a certificate for secure connections under the domain name for use by Learning Center, this doesn't extend to what a workshop may do. If workshop instructions require that you create ingresses in Kubernetes using HTTP only, a `.dev` or `.app` domain name cannot work in the browser.

Note

If you are running Kubernetes on your local machine using a system such as `minikube` and you don't have a custom domain name that maps to the IP address

for the cluster, you can use a `nip.io` address. For example, if `minikube ip` returned `192.168.64.1`, you can use the 192.168.64.1.nip.io domain. You cannot use an address of form `127.0.0.1.nip.io` or `subdomain.localhost`. This causes a failure as internal services needing to connect to each other end up connecting to themselves instead, because the address resolves to the host loopback address of `127.0.0.1`.

If needed, you can override the `shared.ingress_domain` in the values file of Tanzu Application Platform with the `ingressDomain` parameter of learning center:

```
ingressDomain: learningcenter.my-domain.com
```

Set the environment variable manually

Set the `INGRESS_DOMAIN` environment variable on the operator deployment. To set the `INGRESS_DOMAIN` environment variable, run:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_DOMAIN=te
st
```

Where `test` is the domain name for your Kubernetes cluster.

Or if using a `nip.io` address:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_DOMAIN=19
2.168.64.1.nip.io
```

Use of environment variables to configure the operator is a shortcut for a simple use. VMware recommends using Tanzu CLI, or for more complicated scenarios, you can use the `SystemProfile` custom resource.

Enforcing secure connections

By default, the workshop portal and workshop sessions are accessible over HTTP connections. To use secure HTTPS connections, you must have access to a wildcard SSL certificate for the domain under which you want to host the workshops. You cannot use a self-signed certificate.

You can create wildcard certificates by using `letsencrypt <https://letsencrypt.org/>`. After you have the certificate, you can define it as follows.

Configuration YAML

The easiest way to define the certificate is with the configuration passed to Tanzu CLI. So define the `certificate` and `privateKey` properties under the `ingressSecret` property to specify the certificate on the configuration YAML passed to Tanzu CLI:

```
ingressSecret:
  certificate: |
    -----BEGIN CERTIFICATE-----
    MIIFLTCCBBWgAwIBAgaSAys/V2NCTG9uXa9aAiYt7WJ3MA0GCSqGaIb3DQEBCwUA
                                    ...
    dHa6Ly9yMy5vamxlbmNyLm9yZzAiBggrBgEFBQawAoYWaHR0cDoaL3IzLmkubGVu
    -----END CERTIFICATE-----
  privateKey: |
    -----BEGIN PRIVATE KEY-----
    MIIEvQIBADAaBgkqhkiG9waBAQEFAASCBKcwggSjAgEAAoIBAaCx4nyc2xwaVOzf
                                    ...
```

```
    IY/9SatMcJZivH3F1a7SXL98PawPIOSR7986P7rLFHzNjaQQ0DWTaXBRt+oUDxpN
    -----END PRIVATE KEY-----
```

If you already have a TLS secret, follow these steps **before deploying any workshops**:

1. Create the `learningcenter` namespace manually or the one you defined.
2. Copy the TLS secret to the `learningcenter` namespace or to the one you defined, and use the `secretName` property as in this example:

   ```
   ingressSecret:
     secretName: workshops.example.com-tls
   ```

Create the TLS secret manually

To add the certificate as a secret in the `learningcenter` namespace or in the one you defined, the secret must be of type `tls`. You can create it using the `kubectl create secret tls` command:

```
kubectl create secret tls -n learningcenter workshops.example.com-tls --cert=workshop
s.example.com/fullchain.pem --key=workshops.example.com/privkey.pem
```

Having created the secret, if it is the secret corresponding to the default ingress domain you specified earlier, set the `INGRESS_SECRET` environment variable. This way you do not use the configuration passed to Tanzu CLI on the operator deployment. This ensures the secret is applied automatically to any ingress created:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_SECRET=wo
rkshops.example.com-tls
```

If the certificate isn't that of the default ingress domain, you can supply the domain name and name of the secret when creating a workshop environment or training portal. In either case, you must create secrets for the wildcard certificates in the `learningcenter` namespace or the one that you defined.

Specifying the ingress class

Any ingress routes created use the default ingress class. If you have multiple ingress class types available, and you must override which is used, you can define the `ingressClass` property on the configuration YAML as follows.

Configuration YAML

Define the `ingressClass` property on the configuration YAML passed to Tanzu CLI:

```
ingressClass: contour
```

Set the environment variable manually

Set the `INGRESS_CLASS` environment variable for the learningcenter operator:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_CLASS=con
tour
```

This applies only to the ingress created for the training portal and workshop sessions. It does not apply to any ingress created from a workshop as part of the workshop instructions.

This can be necessary when a specific ingress provider is not reliable in maintaining websocket connections. For example, in the case of the nginx ingress controller when there are frequent

creation or deletions of ingresses occurring in the cluster. See the earlier section, Installing and setting up Learning Center operator.

Trusting unsecured registries

One of the options available for workshops is to automatically deploy a container image registry each workshop session. When the Learning Center operator is configured to use a secure ingress with a valid wildcard certificate, the image registry works out of the box.

If the Learning Center operator is not set up to use secure ingress, the image registry is accessed over HTTP and is regarded as not secure.

When using the optional support for building container images using `docker`, the docker daemon deployed for the workshop session is configured for the image registry being not secure yet pushing images to the image registry still works.

In this case of an image registry that is not secure, deploying images from the image registry to the Kubernetes cluster does not work unless the Kubernetes cluster is configured to trust the registry that is not secure.

How you configure a Kubernetes cluster to trust an unsecured registry varies based on how the Kubernetes cluster is deployed and what container runtime it uses.

If you are using `minikube` with `dockerd`, to ensure that the registry is trusted, you must set up the trust the first time you create the minikube instance.

To do this, first determine which IP subnet minikube uses for the inbound ingress router of the cluster. If you already have a minikube instance running, you can determine this by running `minikube ip`. If, for example, this reported `192.168.64.1`, the subnet used is `129.168.64.0/24`.

With this information, when you create a fresh `minikube` instance, you must supply the `--insecure-registry` option with the subnet:

```
minikube start --insecure-registry="129.168.64.0/24"
```

This option tells `dockerd` to regard as not secure any image registry deployed in the Kubernetes cluster and accessed through a URL exposed using an ingress route of the cluster itself.

Currently, there is no way to configure `containerd` to treat as not secure image registries that match a wildcard subdomain or reside in a subnet. It is therefore not possible to run workshops that must deploy images from the per session image registry when using `containerd` as the underlying Kubernetes cluster container runtime. This is a limitation of `containerd`, and there are no known plans for `containerd` to support this ability. This limits your ability to use Kubernetes clusters deployed with a tool such as `kind`, which relies on using `containerd`.

Get started with Learning Center workshops

This topic helps you to get started working with Learning Center workshops. Workshops are where you create your content. You can create a workshop for individual use or group multiple workshops together with a Training Portal.

For more detailed instructions, go to Working with Learning Center Workshops

Creating the workshop environment

With the definition of a workshop already in existence, the first step to deploying a workshop is to create the workshop environment.

To create the workshop environment run:

```
kubectl apply -f {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals/main/resources/workshop-environment.yaml
```

This results in a custom resource being created called `WorkshopEnvironment`:

```
workshopenvironment.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals created
```

The custom resource created is cluster-scoped, and the command needs to be run as a cluster admin or other appropriate user with permission to create the resource.

The Learning Center Operator reacts to the creation of this custom resource and initializes the workshop environment.

For each distinct workshop environment, a separate namespace is created. This namespace is used to hold the workshop instances. The namespace may also be used to provision any shared application services the workshop definition describes which would be used across all workshop instances. Such shared application services are automatically provisioned by the Learning Center Operator when the workshop environment is created.

You can list the workshop environments which have been created by running:

```
kubectl get workshopenvironments
```

This results in the output:

```
NAME                    NAMESPACE              WORKSHOP               IMAGE
URL
lab-k8s-fundamentals   lab-k8s-fundamentals   lab-k8s-fundamentals   {YOUR-REGISTRY-URL}/lab-k8s-fundamentals:main   {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals
```

Additional fields give the name of the workshop environment, the namespace created for the workshop environment, and the name of the workshop the environment was created from.

Requesting a workshop instance

To request a workshop instance, a custom resource of type `WorkshopRequest` needs to be created.

This is a namespaced resource allowing who can create them to be delegated using role-based access controls. Further, in order to be able to request an instance of a specific workshop, you need to know the secret token specified in the description of the workshop environment. If necessary, raising requests against a specific workshop environment can also be constrained to a specific set of namespaces on top of any defined role-based access control (RBAC) rules.

In the context of an appropriate namespace, run:

```
kubectl apply -f {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals/main/resources/workshop-request.yaml
```

This should result in the output:

```
workshoprequest.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals created
```

You can list the workshop requests in a namespace by running:

```
kubectl get workshoprequests
```

This displays output similar to:

```
NAME                    URL                                          USERNAME    PASSWORD
lab-k8s-fundamentals    http://lab-k8s-fundamentals-cvh51.test   learningcenter     buQ
OgZvfHM7m
```

The additional fields provide the URL where the workshop instance can be accessed and the username and password for you to provide when prompted by your web browser.

The user name and password only come into play when you use the lower-level resources to set up workshops. If you use the `TrainingPortal` custom resource, you will see that these fields are empty. This is because, for that case, the workshop instances are deployed so that they rely on user registration and access mediated by the web-based training portal. Visiting the URL for a workshop instance directly when using `TrainingPortal`, redirects you back to the web portal in order to log in if necessary.

You can monitor the progress of this workshop deployment by listing the deployments in the namespace created for the workshop environment:

```
kubectl get all -n lab-k8s-fundamentals
```

For each workshop instance a separate namespace is created for the session. This is linked to the workshop instance, and is where any applications are deployed as part of the workshop. If the definition of the workshop includes a set of resources that should be automatically created for each session namespace, they are created by the Learning Center Operator. It is therefore possible to pre-deploy applications for each session.

In this case, we used `WorkshopRequest`; whereas when using `TrainingPortal`, we created a `WorkshopSession`. The workshop request does result in creating a `WorkshopSession`, but `TrainingPortal` skips the workshop request and directly creates a `WorkshopSession`.

The purpose of having `WorkshopRequest` as a separate custom resource is to allow RBAC and other controls to be used to allow non-cluster administrators to create workshop instances.

Deleting the workshop instance

When you have finished with the workshop instance, you can delete it by deleting the custom resource for the workshop request:

```
kubectl delete workshoprequest/lab-k8s-fundamentals
```

Deleting the workshop environment

If you want to delete the whole workshop environment, it is recommended to first delete all workshop instances. Once this has been done, you can then delete the custom resource for the workshop environment:

```
kubectl delete workshopenvironment/lab-k8s-fundamentals
```

If you don’t delete the custom resources for the workshop requests, the workshop instances are still cleaned up and removed when the workshop environment is removed. The custom resources for the workshop requests still remain, however, and need to be deleted separately.

Get started with Learning Center training portals

This topic describes how you configure and use a `TrainingPortal`, which deploys a set of workshops for attendees.

Working with multiple workshops

The quickest way to deploy a set of workshops to use in a training session is to deploy a `TrainingPortal`. This deploys a set of workshops with one instance of each workshop for each attendee. A web-based portal is provided for registering attendees and allocating them to workshops.

The `TrainingPortal` custom resource provides a high-level mechanism for creating a set of workshop environments and populating it with workshop instances. When the Learning Center operator processes this custom resource, it creates other custom resources to trigger the creation of the workshop environment and the workshop instances. If you want more control, you can use these latter custom resources directly instead.

Loading the workshop definition

A custom resource of type `Workshop` describes each workshop. Before you can create a workshop environment, you must load the definition of the workshop.

Here is an example `Workshop` custom resource:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-k8s-fundamentals
spec:
  title: Kubernetes Fundamentals
  description: Workshop on getting started with Kubernetes
  url: {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals
  vendor: learningcenter.io
  authors:
  - Graham Dumpleton
  difficulty: intermediate
  duration: 1h
  tags:
  - kubernetes
  content:
    image: projects.registry.vmware.com/learningcenter/lab-k8s-fundamentals:latest
  session:
    namespaces:
      budget: medium
    applications:
      terminal:
        enabled: true
        layout: split
      console:
        enabled: true
      editor:
        enabled: true
```

To load the definition of the workshop, run:

```
kubectl apply -f {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals/main/resources/workshop.yaml
```

The custom resource created is cluster-scoped. The command must be run as a cluster admin or other appropriate user with permission to create the resource.

If successfully loaded, the command outputs:

```
workshop.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals created
```

To list the workshop definitions that have been loaded and that can be deployed, run:

```
kubectl get workshops
```

For this workshop, this outputs:

```
NAME                  IMAGE                                                FILES  URL
lab-k8s-fundamentals  {YOUR-REGISTRY-URL}/lab-k8s-fundamentals:main         {YOUR-GIT-
REPO-URL}/lab-k8s-fundamentals
```

The added fields in this case give:

- The name of the custom workshop container image deployed for the workshop.
- A URL for more information about the workshop.

The definition of a workshop is loaded as a step of its own, rather than referring to a remotely hosted definition. This allows a cluster admin to audit the workshop definition to ensure it isn't doing something the cluster admin doesn't want to allow. After the cluster admin approves the workshop definition, it can be used to create instances of the workshop.

Creating the workshop training portal

To deploy a workshop for one or more users, use the `TrainingPortal` custom resource. This custom resource specifies a set of workshops to be deployed and the number of people taking the workshops.

The `TrainingPortal` custom resource we use in this example is:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-k8s-fundamentals
spec:
  workshops:
  - name: lab-k8s-fundamentals
    capacity: 3
    reserved: 1
    expires: 1h
    orphaned: 5m
```

To create the custom resource, run:

```
kubectl apply -f {YOUR-GIT-REPO-URL}/lab-k8s-fundamentals/main/resources/training-port
al.yaml
```

The custom resource created is cluster-scoped. The command must be run as a cluster admin or other appropriate user with permission to create the resource.

This results in the output:

```
trainingportal.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals created
```

There is actually much more going on than this. To see all the resources created, run:

```
kubectl get learningcenter-training -o name
```

You should see:

```
workshop.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals
trainingportal.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals
workshopenvironment.learningcenter.tanzu.vmware.comlab-k8s-fundamentals-w01
workshopsession.learningcenter.tanzu.vmware.com/lab-k8s-fundamentals-w01-s001
```

In addition to the original `Workshop` custom resource providing the definition of the workshop, and the `TrainingPortal` custom resource you just created, you've also created the `WorkshopEnvironment` and `WorkshopSession` custom resources.

The `WorkshopEnvironment` custom resource sets up the environment for a workshop, including deploying any application services that must exist and are shared by all workshop instances.

The `WorkshopSession` custom resource results in the creation of a single workshop instance.

To see a list of the workshop instances created and their details, run:

```
kubectl get workshopsessions
```

This yields output similar to:

```
NAME                            URL                                          USERNAME
PASSWORD
lab-k8s-fundamentals-w01-s001   http://lab-k8s-fundamentals-w01-s001.test
```

Only one workshop instance is created. Though the maximum capacity is set to three, the reserved number of instances (hot spares) is defined as one. Additional workshops instances are only created as workshop sessions are allocated to users. One reserved instance is always maintained until capacity is reached.

If you need a different number of workshop instances, set the `portal.capacity` field of the `TrainingPortal` custom resource YAML input file before creating the resource. Changing the values after the resource is created has no effect.

In this case, only one workshop is listed to be hosted by the training portal. You can deploy more than one workshop at the same time by adding the names of other workshops to `workshops`.

The first time you deploy the workshop, it can take a few moments to pull down the workshop image and start.

To access the workshops, attendees of a training session need to visit the web-based portal for the training session. Find the URL for the web portal by running:

```
kubectl get trainingportals
```

This should yield output similar to:

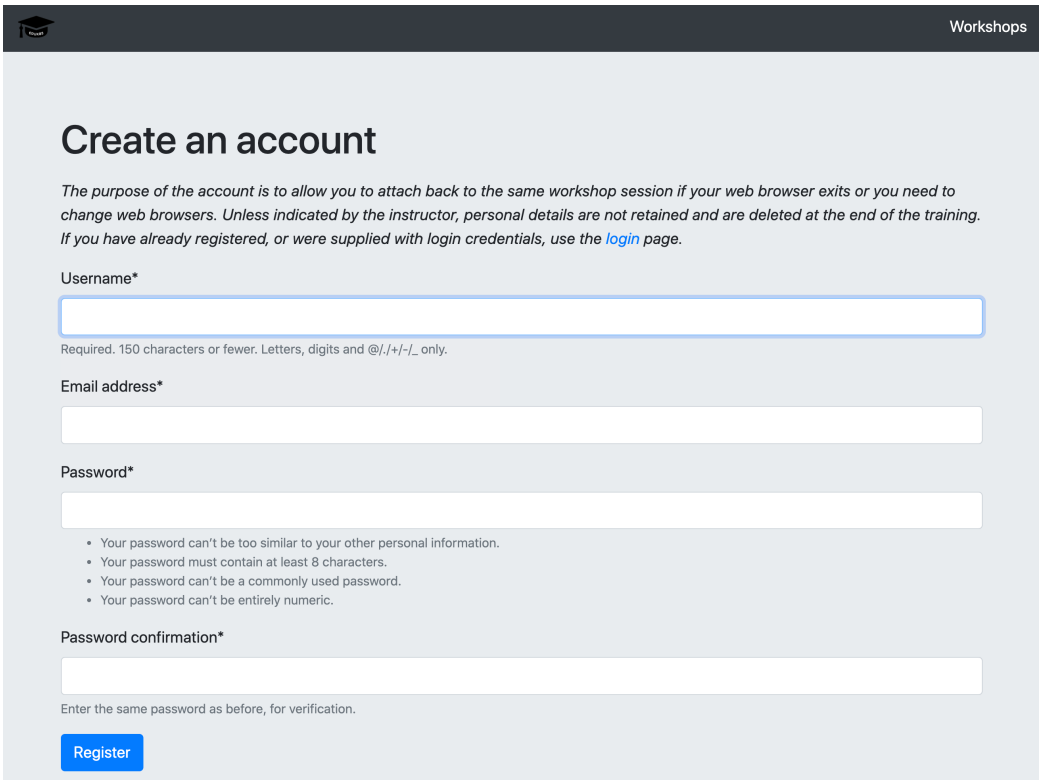
```
NAME                 URL                                    ADMINUSERNAME  ADMINPASSWO
RD
lab-k8s-fundamentals  https://lab-k8s-fundamentals-ui.test  learningcenter         mGI
2C1TkHEBoFgKiZetxMnwAldRU80aN
```

Attendees should only be given the URL. The password listed is only for use by the instructor of the training session if required.

Accessing workshops via the web portal

Attendees can access workshops through the web portal by following two steps:

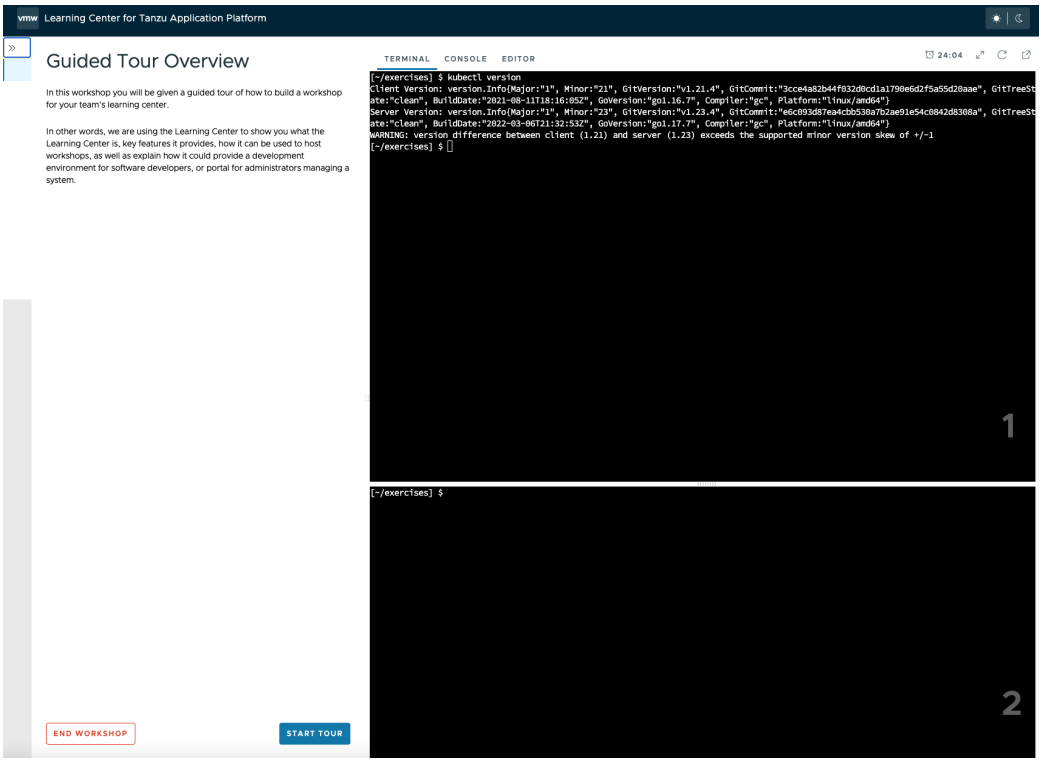
1. The attendee visits the web-based portal for the training session and is presented with a login page. However, before logging in, the attendee must register for an account. The attendee clicks the link to the registration page and fills it in.



Registration is required so if the attendee's web browser exits or the attendee needs to switch web browsers, the attendee can log in again and access the same workshop instance.

2. Upon registering, the attendee is presented with a list of workshops available for the training session.
    - An orange dot beside a workshop means that no instance for that workshop has been allocated to the user as yet, but that some are available.
    - A red dot indicates there are no more workshop instances available.
    - A green dot indicates a workshop instance has already been reserved by the attendee.

    The attendee clicks the "Start workshop" button. This allocates a workshop instance if one hasn't yet been reserved and redirects the attendee to that workshop instance.



Deleting the workshop training portal

The workshop training portal is intended for running workshops with a fixed time period where all workshop instances are deleted when complete.

To delete all workshop instances and the web-based portal, run:

```
kubectl delete trainingportal/lab-k8s-fundamentals
```

Delete Learning Center

This topic describes how you can delete Learning Center.

1. Delete all current workshop environments by running:

   ```
   kubectl delete workshops,trainingportals,workshoprequests,workshopsessions,workshopenvironments --all
   ```

   Ensure the Learning Center operator is still running when running this command.

2. Verify you have deleted all current workshop environments by running:

   ```
   kubectl get workshops,trainingportals,workshoprequests,workshopsessions,workshopenvironments --all-namespaces
   ```

   This command does not delete the workshops in the `workshops.learningcenter.tanzu.vmware.com` package.

3. Uninstall the Learning Center package by running:

   ```
   tanzu package installed delete {NAME_OF_THE_PACKAGE} -n tap-install
   ```

   This command also removes the added custom resource definitions and the `learningcenter` namespace.

> **Note**
>
> If you have installed the Tanzu Application Platform package, Learning Center will be recreated.

4. To remove the Learning Center package, add the following lines to your `tap-values` file.

```
excluded_packages:
- learningcenter.tanzu.vmware.com
- workshops.learningcenter.tanzu.vmware.com
```

Local install guides

The following topics describe how you install Learning Center on your local environment:

- Install on Kind
- Install on Minikube

Local install guides

The following topics describe how you install Learning Center on your local environment:

- Install on Kind
- Install on Minikube

Install Learning Center on Kind

This topic describes how you install Learning Center on your local machine with Kind.

Kind was developed as a means to support development and testing of Kubernetes. Though it exists primarily for that purpose, Kind clusters are often used for local development of user applications as well. For Learning Center, you can use a local Kind cluster to develop workshop content or self-learning when deploying other people's workshops.

Because you are deploying to a local machine, you are unlikely to have access to your own custom domain name and certificate you can use with the cluster. If you don't, you can be restricted as to the sorts of workshops you can develop or run using the Learning Center in Kind. Kind uses `containerd`, which lacks certain features that allow you to trust any image registries hosted within a subnet. This means you cannot readily run workshops that use a local container image registry for each workshop session. If you must run workshops on your local computer that uses an image registry for each session, VMware recommends you use minikube with `dockerd` instead. For more information, see Install on Minikube.

Also, since Kind has limited memory resources available, you may be prohibited from running workshops that have large memory requirements. Workshops that demonstrate the use of third-party applications requiring a multinode cluster also do not work unless the Kind cluster is specifically configured to be multinode rather than single node.

Requirements and setup instructions specific to Kind are detailed in this document. Otherwise, follow normal installation instructions for the Learning Center operator.

Prerequisites

You must complete the following installation prerequisites as a user prior to installation:

- Create a VMware Tanzu Network account and have access to your Tanzu Network credentials.
- Install Kind on your local machine.
- Install Tanzu CLI on your local machine.
- Install Kubernetes command-line tool (kubectl) on your local machine.

Kind cluster creation

When initially creating the Kind cluster, you must configure it so that the ingress controller is exposed. The Kind documentation provides the following command to do this, but check the documentation in case the details have changed.

```
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
EOF
```

Once you have the Kind cluster up and running, you must install an ingress controller.

Ingress controller with DNS

The Kind documentation provides instructions for installing Ambassador, Contour, and Nginx-based ingress controllers.

VMware recommends that you use Contour rather than Nginx, because Nginx drops websocket connections whenever new ingresses are created. The Learning Center workshop environments do include a workaround to re-establish websocket connections for the workshop terminals without losing terminal state, but other applications used with workshops might not, such as terminals available through Visual Studio Code.

Avoid using the Ambassador ingress controller, because it requires all ingresses created to be annotated explicitly with an ingress class of “ambassador.” The Learning Center operator can be configured to do this automatically for ingresses created for the training portal and workshop sessions. However, any workshops that create ingresses as part of the workshop instructions do not work unless they are written to have the user manually add the ingress class when required due to the use of Ambassador.

If you have created a contour ingress controller, verify all pods have a running status. Run:

```
kubectl get pods -n projectcontour -o wide
```

For information about installing Contour, which comes with Tanzu Application Platform, see Install cert-manager, Contour.

Install carvel tools

You must install the kapp controller and secret-gen controller carvel tools in order to properly install VMware tanzu packages.

To install kapp controller, run:

```
kapp deploy -a kc -f https://github.com/vmware-tanzu/carvel-kapp-controller/releases/latest/download/release.yml
```

To install secret-gen controller, run:

```
kapp deploy -a sg -f https://github.com/vmware-tanzu/carvel-secretgen-controller/releases/latest/download/release.yml
```

Note

Type “y” and enter to continue when prompted during installation of both kapp and secret-gen controllers.

Install Tanzu package repository

Follow these steps to install the Tanzu package repository:

1. To create a namespace, run:

   ```
   kubectl create ns tap-install
   ```

2. Create a registry secret:

   ```
   tanzu secret registry add tap-registry \
   --username "TANZU-NET-USER" --password "TANZU-NET-PASSWORD" \
   --server registry.tanzu.vmware.com \
   --export-to-all-namespaces --yes --namespace tap-install
   ```

   Where:

   - `TANZU-NET-USER` and `TANZU-NET-PASSWORD` are your credentials for Tanzu Network.

3. Add a vpackage repository to your cluster:

   ```
   tanzu package repository add tanzu-tap-repository \
   --url registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:VERSION-NUMBER \
   --namespace tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.

   Note

   We are currently on build 7. If this changes, we need to update the command with the correct build version after the –url flag.

4. To check the package repository install status, run:

   ```
   tanzu package repository get tanzu-tap-repository --namespace tap-install
   ```

Wait for a reconciled successful status before attempting to install any other packages.

Create a configuration YAML file for Learning Center package

To create a configuration YAML file:

See Supported yaml file configurations to see a list of configurations you can provide to Learning Center.

1. Create a file called learningcenter-value.yaml in your current directory with the following data:

```
ingressDomain: workshops.example.com
```

Where:

- `ingressDomain` is `<your-local-ip>.nip.io` if you are using a `nip.io` DNS address. Details about this are provided in the following section.
- `workshops.example.com with` is `<your-local-ip>.nip.io`.

Using a `nip.io` DNS address

Before you can start deploying workshops, you must configure the operator to tell it what domain name can be used to access anything deployed by the operator.

Being a local cluster that isn't exposed to the Internet with its own custom domain name, you can use a nip.io. address.

To calculate the `nip.io` address to use, first work out the IP address for the ingress controller exposed by Kind. This is usually the IP address of the local machine itself, even when you use Docker for Mac.

How you get the IP address for your local machine depends on the operating system you are using.

For example on a Mac, you can find your IP address by searching for network using spotlight and selecting the network option under system preferences. Here you can see your IP address under status.

After you have the IP address, add this as a prefix to the domain name `nip.io`. For example, if the address was `192.168.1.1`, use the domain name of `192.168.1.1.nip.io`.

To configure the Learning Center operator with this cluster domain, run:

```
kubectl set env deployment/learningcenter-operator -n eduk8s INGRESS_DOMAIN=192.168.1.
1.nip.io
```

This causes the Learning Center operator to redeploy with the new configuration. You can now deploy workshops.

Note

Some home Internet gateways implement what is called rebind protection. These gateways do not allow DNS names from the public Internet bind to local IP address ranges inside the home network. If your home Internet gateway has such a feature and it is enabled, it blocks `nip.io` addresses from working. In this case, you must configure your home Internet gateway to allow `*.nip.io` names to be bound to local addresses. Also, you cannot use an address of form `127.0.0.1.nip.io` or `subdomain.localhost`. This causes a failure, because when internal services need to

connect to each other, they connect to themselves instead. This happens because the address resolves to the host loopback address of `127.0.0.1`.

Install Learning Center package onto a Kubernetes cluster

To install Learning Center on a Kubernetes cluster:

```
tanzu package install learningcenter --package-name learningcenter.tanzu.vmware.com --
version 0.1.0 -f ./learningcenter-value.yaml --namespace tap-install
```

This package installation uses the installed Package repository with a configuration learningcenter-value.yaml to install our Learning Center package.

Install workshop tutorial package onto a Kubernetes cluster

To install a workshop tutorial on a Kubernetes cluster:

```
tanzu package install learningcenter-tutorials --package-name workshops.learningcente
r.tanzu.vmware.com --version 0.1.0 --namespace tap-install
```

Make sure you install the workshop package after the Learning Center package has reconciled and successfully installed onto your cluster. In case of new versioning, to obtain package version numbers, run:

```
kubectl get packages -n tap-install
```

Run the workshop

To get the training portal URL, run:

```
kubectl get trainingportals
```

You get a URL that you can paste into your browser.

Congratulations, you are now running our tutorial workshop using the Learning Center operator.

Trusting insecure registries

Workshops can optionally deploy a container image registry for a workshop session. This image registry is secured with a password specific to the workshop session and is exposed through a Kubernetes ingress so it can be accessed from the workshop session.

In a typical scenario, Kind uses insecure ingress routes. Even were you to generate a self-signed certificate to use for ingress, it is not trusted by `containerd` that runs within Kind. You must tell Kind to trust any insecure registry running inside of Kind.

You must configure Kind to trust insecure registries when you first create the cluster. Kind, however, is that it uses `containerd` and not `dockerd`. The `containerd` runtime doesn't provide a way to trust any insecure registry hosted within the IP subnet used by the Kubernetes cluster. Instead, `containerd` requires that you enumerate every single host name or IP address on which an insecure registry is hosted. Because each workshop session created by the Learning Center for a workshop uses a different host name, this becomes cumbersome.

If you must used Kind, find out the image registry host name for a workshop deployment and configure `containerd` to trust a set of host names corresponding to low-numbered sessions for that workshop. This allows Kind to work, but once the host names for sessions go beyond the range of

host names you set up, you need to delete the training portal and recreate it, so you can use the same host names again.

For example, if the host name for the image registry were of the form:

```
lab-docker-testing-wMM-sNNN-registry.192.168.1.1.nip.io
```

where `NNN` changes per session, you must use a command to create the Kind cluster. For example:

```
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
containerdConfigPatches:
- |
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s001-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s001-registry.192.168.1.1.nip.io"]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s002-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s002-registry.192.168.1.1.nip.io"]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s003-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s003-registry.192.168.1.1.nip.io"]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s004-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s004-registry.192.168.1.1.nip.io"]
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."lab-docker-testing-w01-s005-r
egistry.192.168.1.1.nip.io"]
    endpoint = ["http://lab-docker-testing-w01-s005-registry.192.168.1.1.nip.io"]
EOF
```

This allows you to run five workshop sessions before you have to delete the training portal and recreate it.

If you use this, you can use the feature of the training portal to automatically update when a workshop definition is changed. This is because the `wMM` value identifying the workshop environment changes any time you update the workshop definition.

There is no other known workaround for this limitation of `containerd`. As such, VMware recommends you use minikube with `dockerd` instead. For more information, see Install on Minikube.

Install Learning Center on Minikube

This topic describes how you install Learning Center on your local machine with Minikube.

Minikube enables local deployment of Kubernetes for developing workshop content or for self-learning when deploying other people's workshops.

Because you are deploying to a local machine, you are unlikely to have access to your own custom domain name and certificate you can use with the cluster. You must take extra steps over a standard install of Minikube to ensure you can run certain types of workshops.

Also, because Minikube generally has limited memory resources available and is only a single-node cluster, you might be restricted from running workshops that have large memory requirements or that demonstrate the use of third-party applications requiring a multinode cluster.

Requirements and setup instructions specific to Minikube are detailed in this document. Otherwise, you can follow normal installation instructions for the Learning Center operator.

Trusting insecure registries

Workshops can optionally deploy a container image registry for a workshop session. This image registry is secured with a password specific to the workshop session and is exposed through a Kubernetes ingress so it can be accessed from the workshop session.

In a typical scenario, Minikube uses insecure ingress routes. Even were you to generate a self-signed certificate to use for ingress, it is not trusted by `dockerd` that runs within Minikube. You must tell Minikube to trust any insecure registry running inside of Minikube.

You must configure Minikube to trust insecure registries the first time you start a new cluster with it. That is, you must supply the details to `minikube start`, which means you must know the IP subnet Minikube uses.

If you already have a cluster running using Minikube, run `minikube ip` to discover the IP address it uses. From that you can discover the trusted subnet. For example, if `minikube ip` returned `192.168.64.1`, the trusted subnet is `192.168.64.0/24`.

With this information, when you start a new cluster with Minikube, run:

```
minikube start --insecure-registry=192.168.64.0/24
```

If you already have a cluster started with Minikube, you cannot stop it and then provide this option when it is restarted. You can only use this option for a completely new cluster.

You must also use `dockerd`, not `containerd`, in the Minikube cluster. `containerd` does not accept an IP subnet when defining insecure registries to be trusted. It allows only specific hosts or IP addresses. Because you don't know what IP address Minikube will use in advance, you can't provide the IP address on the command line when starting Minikube to create the cluster.

Prerequisites

You must complete the following installation prerequisites as a user prior to installation:

- Create a tanzunet account and have access to your tanzunet credentials.
- Install miniKube on your local machine.
- Install tanzuCLI on your local machine.
- Install kubectlCLI on your local machine.

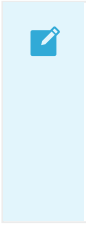
Ingress controller with DNS

After the Minikube cluster is running, you must enable the `ingress` and `ingress-dns` add-ons for Minikube. These deploy the nginx ingress controller along with support for integrating into DNS.

To enable these after the cluster has been created, run:

```
minikube addons enable ingress
minikube addons enable ingress-dns
```

You are now ready to install the Learning Center package.



Note

The ingress add-ons for Minikube do not work when using Minikube on top of Docker for Mac or Docker for Windows. On macOS you must use the Hyperkit VM driver. On Windows you must use the Hyper-V VM driver.

Install carvel tools

You must install the kapp controller and secret-gen controller carvel tools in order to properly install VMware tanzu packages.

To install kapp controller, run:

```
kapp deploy -a kc -f https://github.com/vmware-tanzu/carvel-kapp-controller/releases/latest/download/release.yml
```

To install secret-gen controller, run:

```
kapp deploy -a sg -f https://github.com/vmware-tanzu/carvel-secretgen-controller/releases/latest/download/release.yml
```

Type “y” and enter to continue when prompted during installation of both kapp and secret-gen controllers.

Install Tanzu package repository

Follow these steps to install the Tanzu package repository:

1. To create a namespace, run:

   ```
   kubectl create ns tap-install
   ```

2. Create a registry secret:

   ```
   tanzu secret registry add tap-registry \
     --username "TANZU-NET-USER" --password "TANZU-NET-PASSWORD" \
     --server registry.tanzu.vmware.com \
     --export-to-all-namespaces --yes --namespace tap-install
   ```

   Where:

   - `TANZU-NET-USER` and `TANZU-NET-PASSWORD` are your credentials for Tanzu Network.

3. Add a package repository to your cluster:

   ```
   tanzu package repository add tanzu-tap-repository \
     --url registry.tanzu.vmware.com/tanzu-application-platform/tap-packages:VERSION-NUMBER \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.

   Note

> We are currently on build 7; if this changes, we need to update the command with the correct build version after the –url flag.

4. To check the package repository install status, run:

```
tanzu package repository get tanzu-tap-repository --namespace tap-install
```

Wait for a reconciled sucessful status before attempting to install any other packages.

Create a configuration YAML file for the Learning Center package

Create a file called learningcenter-value.yaml in your current directory with the following data:

See Supported yaml file configurations to see a list of configurations you can provide to Learning Center.

```
ingressDomain: workshops.example.com
```

Where:

- `ingressDomain` is `<your-local-ip>.nip.io` if you are using a `nip.io` DNS address. Details about this are provided in the following section.
- `workshops.example.com` is `<your-local-ip>.nip.io`

Using a `nip.io` DNS address

After the Learning Center operator is installed, before you can start deploying workshops, you must configure the operator to tell it what domain name can be used to access anything deployed by the operator.

Being a local cluster that isn't exposed to the Internet with its own custom domain name, you can use a nip.io. address.

To calculate the `nip.io` address to use, first work out the IP address of the cluster created by Minikube by running `minikube ip`. Add this as a prefix to the domain name `nip.io`. For example, if `minikube ip` returns `192.168.64.1`, use the domain name of `192.168.64.1.nip.io`.

To configure the Learning Center operator with this cluster domain, run:

```
kubectl set env deployment/learningcenter-operator -n learningcenter INGRESS_DOMAIN=19
2.168.64.1.nip.io
```

This causes the Learning Center operator to redeploy with the new configuration. You should now be able to start deploying workshops.

> **Note**
>
> Some home Internet gateways implement what is called rebind protection. These gateways do not let DNS names from the public Internet bind to local IP address ranges inside the home network. If your home Internet gateway has such a feature and it is enabled, it blocks `nip.io` addresses from working. In this case, you must configure your home Internet gateway to allow `*.nip.io` names to be bound to local addresses.

Install Learning Center package onto a minikube cluster

To install the Learning Center package onto a minikube cluster, run:

```
tanzu package install learningcenter --package-name learningcenter.tanzu.vmware.com --
version 0.1.0 -f ./learningcenter-value.yaml --namespace tap-install
```

This package installation uses the installed Package repository with a configuration learningcenter-value.yaml to install the Learning Center package.

Install workshop tutorial package onto a minikube cluster

To install the workshop tutorial package onto a minikube cluster, run:

```
tanzu package install learningcenter-tutorials --package-name workshops.learningcente
r.tanzu.vmware.com --version 0.1.0 --namespace tap-install
```

Make sure you install the workshop package after the Learning Center package has reconciled and successfully installed onto your cluster. In case of new versioning, to obtain package version numbers, run:

```
kubectl get packages -n tap-install
```

Run the workshop

To get the training portal URL, run:

```
kubectl get trainingportals
```

You get a URL that you can paste into your browser.

Congratulations, you are now running the tutorial workshop using the Learning Center operator.

Working with large images

If you create or run workshops that work with the image registry created for a workshop session, and you push images to that image registry that have large layers, you must configure the version of nginx deployed for the ingress controller and increase the allowed size of request data for a HTTP request.

To do this, run:

```
kubectl edit configmap nginx-load-balancer-conf -n kube-system
```

To the config map resource, add the following property under `data`:

```
proxy-body-size: 1g
```

If you don’t increase this, `docker push` fails when trying to push container images with large layers.

Limited resource availability

When deploying a cluster, by default Minikube only configures support for 2Gi of memory. This usually isn’t adequate.

To view how much memory is available when a custom amount has been set as a default, run:

```
minikube config get memory
```

VMware recommends you configure Minikube to use 4Gi or more. This must be specified when the cluster is first created. Do this by using the `--memory` option to `minikube start` or by specifying a default memory value beforehand by using `minikube config set memory`.

In addition to increasing the memory available, you can increase the disk size, because fat container images can quickly use disk space within the cluster.

Storage provisioner issue

v1.12.3 of Minikube introduced a bug in the storage provisioner that causes potential corruption of data in persistent volumes where the same persistent volume claim name is used in two different namespaces. This affects Learning Center when:

- You deploy multiple training portals at the same time.
- You run multiple workshops at the same time that have docker or image registry support enabled.
- The workshop session itself is backed by persistent storage and multiple sessions run at the same time.

This issue is supposed to be fixed in Minikube v1.13.0; however, you can still encounter issues when deleting a training portal instance and recreating it immediately with the same name. This occurs because reclaiming of the persistent volume by the Minikube storage provisioner can be slow, and the new instance can grab the same original directory on disk with old data in it. After deleting a training portal instance, wait before recreating one with the same name to allow the storage provisioner to delete the old persistent volume.

Create workshops for Learning Center

This section provides information about how you create Learning Center workshops.

- Workshop configuration
- Workshop images
- Workshop content
- Build an image
- Workshop instructions
- Workshop runtime
- Workshop slides
- Air-gapped environment requirements

Create workshops for Learning Center

This section provides information about how you create Learning Center workshops.

- Workshop configuration
- Workshop images
- Workshop content
- Build an image
- Workshop instructions

- Workshop runtime
- Workshop slides
- Air-gapped environment requirements

Configure your Learning Center workshop

This topic describes the two main steps required to configure your Learning Center workshop. The first specifies the structure of the workshop content and the second defines the runtime requirements for deploying the workshop.

Specifying structure of the content

There are multiple ways you can configure a workshop to specify the structure of the content. The sample workshops use YAML files.

The `workshop/modules.yaml` file provides details about the list of available modules that make up your workshop and data variables for use in content.

The list of available modules represents all of the modules available to you. You might not use all of them. You might want to run variations of your workshop, such as for different programming languages. As such, which modules are active and are used for a specific workshop are listed in the separate `workshop/workshop.yaml` file. The active modules are listed with the name to be given to that workshop.

By default the `workshop.yaml` file specifies what modules are used. When you want to deliver different variations of the workshop content, you can provide multiple workshop files with different names. For example, you can name the workshop files `workshop-java.yaml` and `workshop-python.yaml`.

Where you have multiple workshop files and don't have the default `workshop.yaml` file, you can specify the default workshop file by setting the `WORKSHOP_FILE` environment variable in the runtime configuration.

The format for listing the available modules in the `workshop/modules.yaml` file is:

```
modules:
  workshop-overview:
    name: Workshop Overview
    exit_sign: Setup Environment
  setup-environment:
    name: Setup Environment
    exit_sign: Start Workshop
  exercises/01-sample-content:
    name: Sample Content
  workshop-summary:
    name: Workshop Summary
    exit_sign: Finish Workshop
```

Each available module is listed under `modules`, where the name used corresponds to the path to the file containing the content for that module. Any extension identifying the content type is left off.

For each module, set the `name` field to the page title to be displayed for that module. If no fields are provided and `name` is not set, the title for the module is derived from the name of the module file.

The corresponding `workshop/workshop.yaml` file, where all available modules are used, would have the format:

```
name: Markdown Sample
modules:
  activate:
    - workshop-overview
    - setup-environment
    - exercises/01-sample-content
    - workshop-summary
```

The top-level `name` field in this file is the name of this variation of the workshop content.

The `modules.activate` field is a list of modules to be used for the workshop. The names in this list must match the names as they appear in the modules file.

The order in which modules are listed under the `modules.activate` field in the workshop configuration file dictates the order pages are traversed. The order in which modules appear in the modules configuration file is not relevant.

At the bottom of each page, a **Continue** button is displayed to allow the user to go to the next page in sequence. You can customize the label on this button by setting the `exit_sign` field in the entry for the module in the modules configuration file.

In the last module in the workshop, a button is displayed, but where the user goes after clicking it varies. If you want the user to go to a different website upon completion, you can set the `exit_link` field of the final module to an external URL. Alternatively, you can set the `RESTART_URL` environment variable in a workshop environment to control where the user goes. If a destination for the final page is not provided, the user is redirected back to the starting page of the workshop.

When the user uses the training portal, the training portal overrides this environment variable so, at the completion of a workshop, the user returns to the training portal.

VMware recommends that for the last page, the `exit_sign` be set to “Finish Workshop” and `exit_link` not be specified. This enables the destination to be controlled from the workshop environment or training portal.

Specifying the runtime configuration

You can deploy workshop images directly to a container runtime. The Learning Center Operator is provided to manage deployments into a Kubernetes cluster. You define the configuration for the Learning Center Operator with a `Workshop` CRD in the `resources/workshop.yaml` file:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  vendor: learningcenter.tanzu.vmware.com
  title: Markdown Sample
  description: A sample workshop using Markdown
  url: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  duration: 15m
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

Where:

- `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

In this sample, a custom workshop image bundles the workshop content into its own container image. The `content.image` setting specifies this. To instead download workshop content from a GitHub repository at runtime, use:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  vendor: learningcenter.tanzu.vmware.com
  title: Markdown Sample
  description: A sample workshop using Markdown
  url: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  content:
    files: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  duration: 15m
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

Where:

- `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

The difference is the use of the `content.files` setting. Here, the workshop content is overlaid on top of the standard workshop base image. To use an alternate base image with additional applications or packages installed, specify the alternate image against the `content.image` setting at the same time you set `content.files`.

Next steps

- Learn about configuration options for the workshop.yaml custom resource definitions (CRD) in Workshop resource.

Create the image for your Learning Center workshop

The workshop environment for the Learning Center is packaged as a container image. This topic describes how you create the Learning Center workshop image.

You can execute the image with remote content pulled down from GitHub or a web server. Alternatively, you can bundle your workshop content, including any extra tools required, in a new container image derived from the workshop environment base image.

Templates for creating a workshop

To get you started with your own workshop content, VMware provides a number of sample workshops. Different templates in Markdown or AsciiDoc are available to use depending on the syntax you use to create the workshop. These templates are available in a zip file called `LEARNING-CENTER-WORKSHOP-SAMPLES.ZIP` on the Tanzu Network TAP Product Page. The zip file contains the following projects that you can upload to your own Git repository:

- lab-markdown-sample
- lab-asciidoc-sample

When creating your own workshops, a suggested convention is to prefix the directory name with the Git repository name where it is hosted. For example, you can make the prefix `lab-`. This way it stands out as a workshop or lab when you have a number of Git repositories on the same Git hosting service account or organization.

Important

Do not make the name you use for a workshop too long. The DNS host name used for applications deployed from the workshop, when using certain methods of deployment, might exceed the 63 character limit. This is because the workshop deployment name is used as part of the namespace for each workshop session. This in turn is used in the DNS host names generated for the ingress host name. VMware suggests keeping the workshop name, and so your repository name, to 25 characters or less.

Workshop content directory layout

After creating a copy of the sample workshop content, you can see a number of files located in the top-level directory and a number of subdirectories forming a hierarchy. The files in the top-level directory are:

- `README.md` - A file stating what the workshop in your Git repository is about and how to deploy it. Replace the current content provided in the sample workshop with your own.
- `LICENSE` - A license file so people are clear about how they can use your workshop content. Replace this with what license you want to apply to your workshop content.
- `Dockerfile` - Steps to build your workshop into an image ready for deployment. Leave this as is, unless you want to customize it to install additional system packages or tools.
- `kustomization.yaml` - A kustomize resource file for loading the workshop definition. The Learning Center operator must be deployed before using this file.
- `.dockerignore` - List of files to ignore when building the workshop content into an image.
- `.eduk8signore` - List of files to ignore when downloading workshop content into the workshop environment at runtime.

Key subdirectories and the files contained within them are:

- `workshop` - Directory under which your workshop files reside.
- `workshop/modules.yaml` - Configuration file with details of available modules that make up your workshop and data variables for use in content.
- `workshop/workshop.yaml` - Configuration file that gives the name of the workshop, the list of active modules for the workshop, and any overrides for data variables.
- `workshop/content` - Directory under which your workshop content resides, including images to be displayed in the content.
- `resources` - Directory under which Kubernetes custom resources are stored for deploying the workshop using the Learning Center.
- `resources/workshop.yaml` - The custom resources for the Learning Center, which describe your workshop and requirements for deployment.

- `resources/training-portal.yaml` - A sample custom resource for the Learning Center for creating a training portal for the workshop, encompassing the workshop environment and a workshop instance.

A workshop can include other configuration files and directories with other types of content, but this is the minimal set of files to get you started.

Directory for workshop exercises

The number of files and directories can quickly add up at the top level of your repository. The same is true of the home directory for the user when running the workshop environment. To help with this proliferation of files, you can push files required for exercises during the workshop into the `exercises` subdirectory under the root of the repository.

With an `exercises` subdirectory, the initial working directory for the embedded terminal when created is set to `$HOME/exercises` instead of `$HOME`. If the embedded editor is enabled, the subdirectory is opened as the workspace for the editor. Only directories and files in that subdirectory are visible through the default view of the editor.

However, the `exercises` directory isn't set as the home directory of the user. This means if a user inadvertently runs `cd` with no arguments from the terminal, they go back to the home directory.

To avoid confusion and help a user return to where they must be, VMware recommends that when you instruct users to change directories, provide a full path relative to the home directory. For example, use a path of the form `~/exercises/example-1` rather than `example-1` for the `cd` command when changing directories. By using a full path, users can execute the command and be assured of going to the required location.

Working on your Learning Center workshop content

This topic tells you about the best practices for speeding up the iterative loop of editing and testing a Learning Center workshop when developing the content.

Workshop content is either embedded in a custom workshop image or downloaded from a Git repository or web server when the workshop session is created.

Deactivating reserved sessions

Deactivate the reserved sessions by setting the `reserved` field to `0` in your training portal instance:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-sample-workshop
spec:
  portal:
    sessions:
      maximum: 1
  workshops:
  - name: lab-sample-workshop
    reserved: 0
    expires: 120m
    orphaned: 15m
```

If you do not deactivate reserved sessions, a new session is always created ready for the next workshop session when there is available capacity to do so. If you modify workshop content while testing the current workshop session, terminate the session and start a new one, the workshop picks up the reserved session. The reserved session has a copy of the old content.

By deactivating reserved sessions, a new workshop session is always created on demand. This ensures the latest workshop content is used.

Because you might have to wait to create a new workshop, shut down the existing workshop session first. The new workshop session might also take some time to start if an updated version of the workshop image also has to be pulled down.

Live updates to the content

If you download workshop content from a Git repository or web server, and you are only doing simple updates to workshop instructions, scripts, or files bundled with the workshop, you can update the content in place without needing to restart the workshop session. To perform an update, download the workshop content after you have pushed back any changes to the hosted Git repository or updated the content available through the web server. From the workshop session terminal, run:

```
update-workshop
```

This command downloads any workshop content from the Git repository or web server, unpacks it into the live workshop session, and re-runs any script files found in the `workshop/setup.d` directory.

Find the location where the workshop content is downloading by viewing the file:

```
cat ~/.eduk8s/workshop-files.txt
```

You can change the location saved in this file if, for example, it references a specific version of the workshop content and you want to test with a different version.

Once the workshop content has been updated, reload the current page of the workshop instructions by clicking the reload icon on the dashboard while holding down the shift key.

If additional pages are added to the workshop instructions or pages are renamed, you must restart the workshop renderer process by running:

```
restart-workshop
```

If you didn't rename the current pager or if the name changed, you can trigger a reload of the current page. Click the home icon or refresh the webpage if the name of the first page didn't change.

If action blocks within the workshop instructions are broken, to change and test the workshop instructions within the live workshop session, you can edit the appropriate page under `/opt/workshop/content`. Navigate to the modified page or reload it to verify the change.

To change set up scripts that create files specific to a workshop session, edit the script under `/opt/workshop/setup.d` directory.

To trigger running of any setup scripts, run:

```
rebuild-workshop
```

If local changes to the workshop session take effect, you can restore the file in the original Git repository.

Updating workshop content in a live session in this way does not undo any deployments or changes you make in the Kubernetes cluster for that session. To retest parts of the workshop instructions, you might have to manually undo the changes in the cluster to replay them. This depends on your specific workshop content.

Custom workshop image changes

If your workshop uses a custom workshop image to provide additional tools and you have included the workshop instructions as part of the workshop image, you must use an image tag of `main`, `develop`, or `latest` during the development of workshop content. Do not use a version image reference.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-sample-workshop
spec:
  title: Sample Workshop
  description: A sample workshop
  content:
    image: {YOUR-GIT-REPO-URL}/lab-sample-workshop:main
```

When you use an image tag of `main`, `develop`, or `latest`, the image pull policy is set to `Always` to ensure that the custom workshop image is pulled down again for a new workshop session if the remote image changes. If the image tag is for a specific version, you must change the workshop definition every time when the workshop image changes.

Custom workshop image overlay

For a custom workshop image, you can set up the workshop definition to pull down the workshop content from the hosted Git repository or web server as the follows:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-sample-workshop
spec:
  title: Sample Workshop
  description: A sample workshop
  content:
    image: {YOUR-REGISTRY-URL}/lab-sample-workshop:main
    files: {YOUR-GIT-REPO-URL}/lab-sample-workshop
```

By pulling down the workshop content as an overlay of the custom workshop image when the workshop session starts, you only need to rebuild the custom workshop image when you need to make changes such as to include additional tools or to ensure the latest workshop instructions are included in the final custom workshop image.

Because the location of the workshop files is known, you can live update the workshop content in the session by following Live updates to the content.

If the additional set of tools required for a workshop is not specific to a workshop, VMware recommends that you create a standalone workshop base image where you can add the tools. You can always pull down content for a specific workshop from a Git repository or web server when the workshop session starts.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-sample-workshop
spec:
  title: Sample Workshop
  description: A sample workshop
```

```
  content:
    image: {YOUR-REGISTRY-URL}/custom-environment:main
    files: {YOUR-GIT-REPO-URL}/lab-sample-workshop
```

This separates generic tooling from specific workshops and so you can use the custom workshop base image for multiple workshops on different, but related topics that require the same tooling.

Changes to workshop definition

By default, to modify the definition for a workshop, you need to delete the training portal instance, update the workshop definition in the cluster, and recreate the training portal.

During the workshop content development, to change resource allocations, role access, or to specify what resource objects to be automatically created for the workshop environment or a specific workshop session, you can enable automatic updates in the training portal definition by setting `updates.workshop` field as `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-sample-workshop
spec:
  portal:
    sessions:
      maximum: 1
    updates:
      workshop: true
  workshops:
  - name: lab-sample-workshop
    expires: 120m
    orphaned: 15m
```

With automatic updates enabled, if the workshop definition in the cluster is modified, the existing workshop environment managed by the training portal for that workshop is shut down and replaced with a new workshop environment by using the updated workshop definition.

When an active workshop session is running, the actual deletion of the old workshop environment is delayed until that workshop session is terminated.

Local build of workshop image

If you do not package a workshop into a custom workshop image, VMware recommends to build a custom workshop image locally on your own machine by using `docker` to avoid keeping pushing changes to a hosted Git repository and using a Kubernetes cluster for local workshop content development.

Furthermore, to avoid pushing the image to a public image registry on the Internet, you must deploy an image registry to your local Kubernetes cluster where you run the Learning Center. In most cases, a basic deployment of an image registry in a local cluster access is not secure. As a result, you have to configure the Kubernetes cluster to trust the registry that is not secure. This can be difficult to do depending on the Kubernetes cluster you use, but it can enable quicker turnaround because you do not have to push or pull the custom workshop image across the public Internet.

After pushing the custom workshop image built locally to the local image registry, you can set the image reference in the workshop definition to pull the custom workshop from the local registry in the same cluster. To ensure that the custom workshop image is always pulled for a new workshop session after update, use the `latest` tag when tagging and pushing the image to the local registry.

Build an image for your Learning Center workshop

This topic describes how you include an extra system, third-party tool, or configuration in your image by bundling workshop content from the Learning Center workshop base image.

The following sample workshop template provides a `Dockerfile`.

Structure of the Dockerfile

The structure of the `Dockerfile` in the sample workshop template is:

```
FROM registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:xxxxxxxx
xxxxxxxxxxxxxx

COPY --chown=1001:0 . /home/eduk8s/

RUN mv /home/eduk8s/workshop /opt/workshop

RUN fix-permissions /home/eduk8s
```

The default `Dockerfile` action is to:

- Copy all files from a registry to the `/home/eduk8s` directory.
    - You must build the custom workshop images on the base environment image according to the version of Tanzu Application Platform. To get the image ID, run:

      ```
      kubectl get ds -n learningcenter learningcenter-prepull -o=jsonpath="{.sp
      ec.template.spec.initContainers[0].image}"
      ```

      Example image ID:

      ```
      registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:
      a8870aa60b45495d298df5b65c69b3d7972608da4367bd6e69d6e392ac969dd4
      ```

    - You can build the workshop images directly on the base environment image, or you can create an intermediate base image to install extra packages required by a number of different workshops.
    - The `--chown=1001:0` option ensures that files are owned by the appropriate user and group.
- The `workshop` subdirectory is moved to `/opt/workshop` so that it is not visible to the user. This subdirectory is in an area searchable for workshop content, in addition to `/home/eduk8s/workshop`.

To customize your `Dockerfile`:

- You can ignore other files or directories from the repository, by listing them in the `.dockerignore` file.
- You can include `RUN` statements in the `Dockerfile` to run custom-build steps, but the `USER` inherited from the base image has user ID `1001` and is not the `root` user.

Custom workshop base images

The `base-environment` workshop images include language run times for Node.js and Python. If you need a different language runtime or a different version of a language runtime, you must create a custom workshop base image which includes the environment you need. This custom workshop image is derived from `base-environment` but includes extra runtime components.

The following Dockerfile example creates a Java JDK11-customized image:

```
ARG IMAGE_REPOSITORY=dev.registry.tanzu.vmware.com/learning-center
FROM ${IMAGE_REPOSITORY}/pkgs-java-tools as java-tools
FROM registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:xxxxxxxx
xxxxxxxxxx
COPY --from=java-tools --chown=1001:0 /opt/jdk11 /opt/java
COPY --from=java-tools --chown=1001:0 /opt/gradle /opt/gradle
COPY --from=java-tools --chown=1001:0 /opt/maven /opt/maven
COPY --from=java-tools --chown=1001:0 /opt/code-server/extensions/.  /opt/code-server/
extensions/
COPY --from=java-tools --chown=1001:0 /home/eduk8s/. /home/eduk8s/
COPY --from=java-tools --chown=1001:0 /opt/eduk8s/. /opt/eduk8s/
ENV PATH=/opt/java/bin:/opt/gradle/bin:/opt/maven/bin:$PATH \
    JAVA_HOME=/opt/java \
    M2_HOME=/opt/maven
```

Installing extra system packages

Installing extra system packages requires that you run the installation as `root`. You must switch the user commands before running the command, and then switch the user back to user ID of `1001`.

```
USER root

RUN ... commands to install system packages

USER 1001
```

VMware recommends that you only use the `root` user to install extra system packages. Don't use the `root` user when adding anything under `/home/eduk8s`. Otherwise, you must ensure the user ID and group for directories and files are set to `1001:0` and then run the `fix-permissions` command if necessary.

When you run any command as `root`, you must temporarily override the value of the `HOME` environment variable and set it to `/root`.

If you don't do this the `root` user drops configuration files in `/home/eduk8s`, thinking it is the `root` home directory, because the `HOME` environment variable is by default set to `/home/eduk8s`. This can cause commands run later during the workshop to fail if they try to update the configuration files as they have wrong permissions.

Fixing the file and group ownership and running `fix-permissions` can help with this problem, but not in every case, because of permissions the `root` user may apply and how container image layers work. VMware recommends that you use the following:

```
USER root

RUN HOME=/root && \
    ... commands to install system packages

USER 1001
```

Installing third-party packages

If you are not using system packaging tools to install extra packages, but are manually downloading packages and optionally compiling them to binaries, it is better to do this as the default user and not `root`.

If compiling packages, VMware recommends working in a temporary directory under `/tmp` and removing the directory as part of the same `RUN` statement when done.

If you are installing a binary, you can install it in `/home/eduk8s/bin`. This directory is in the application search path defined by the `PATH` environment variable for the image.

To install a directory hierarchy of files, create a separate directory under `/opt` to install everything. You can override the `PATH` environment variable in the `Dockerfile` to add an extra directory for application binaries and scripts. You can override the `LD_LIBRARY_PATH` environment variable for the location of shared libraries.

If installing any files from a `RUN` instruction into `/home/eduk8s`, VMware recommends that you run `fix-permissions` as part of the same instruction to avoid copies of files being made into a new layer, which applies to the case where `fix-permissions` is only run in a later `RUN` instruction. You can still leave the final `RUN` instruction for `fix-permissions` as it is smart enough not to apply changes if the file permissions are already set correctly and so it does not trigger a copy of a file when run more than once.

Writing instructions for your Learning Center workshop

This topic describes how you write and format the instructions for a Learning Center workshop. You can use either Markdown with file extension `.md` or AsciiDoc with file extension `.adoc` as the markup format for the individual module files that comprise the workshop instructions.

Annotation of executable commands

In conjunction with the standard Markdown and AsciiDoc, you can apply additional annotations to code blocks. The annotations indicate that a user can click the code block and have it copied to the terminal and executed.

If using Markdown, to annotate a code block so it is copied to the terminal and executed, use:

```
```execute
echo "Execute command."
```
```

When the user clicks the code block, the command is executed in the first terminal of the workshop dashboard.

If using AsciiDoc, you can instead use the `role` annotation in an existing code block:

```
[source,bash,role=execute]
----
echo "Execute command."
----
```

When the workshop dashboard is configured to display multiple terminals, you can qualify which terminal the command must be executed in by adding a suffix to the `execute` annotation. For the first terminal, use `execute-1`, for the second terminal `execute-2`, and so on:

```
```execute-1
echo "Execute command."
```

```execute-2
echo "Execute command."
```
```

To execute a command in all terminal sessions on the terminals tab of the dashboard, you can use `execute-all`:

```
```execute-all
clear
```
```

In most cases, a command the user executes completes immediately. To run a command that never returns, with the user needing to interrupt it to stop it, you can use the special string `<ctrl+c>` in a subsequent code block.

```
```execute
<ctrl+c>
```
```

When the user clicks on this code block, the command running in the corresponding terminal is interrupted.

Note

Using the special string `<ctrl+c>` is deprecated, and you must use the `terminal:interrupt` clickable action instead.

Annotation of text to be copied

To copy the content of the code block into the paste buffer instead of running the command, you can use:

```
```copy
echo "Text to copy."
```
```

After the user clicks this code block, they can then paste the content into another window.

If you have a situation where the text being copied must be modified before use, you can denote this special case by using `copy-and-edit` instead of `copy`. The text is still copied to the paste buffer, but is displayed in the browser in a way to highlight that it must be changed before use.

```
```copy-and-edit
echo "Text to copy and edit."
```
```

For AsciiDoc, similar to `execute`, you can add the `role` of `copy` or `copy-and-edit`:

```
[source,bash,role=copy]
----
echo "Text to copy."
----

[source,bash,role=copy-and-edit]
----
echo "Text to copy and edit."
----
```

For `copy` only, to mark an inline code section within a paragraph of text as copyable when clicked, you can append the special data variable reference `{{copy}}` immediately after the inline code block:

```
Text to `copy`{{copy}}.
```

Extensible clickable actions

The preceding means to annotate code blocks were the original methods used to indicate code blocks to be executed or copied when clicked. To support a growing number of clickable actions with different customizable purposes, annotation names are now name-spaced. The preceding annotations are still supported, but the following are now recommended, with additional options available to customize the way the actions are presented.

For code execution, instead of:

```
```execute
echo "Execute command."
```
```

you can use:

```
```terminal:execute
command: echo "Execute command."
```
```

The contents of the code block is YAML. The executable command must be set as the `command` property. By default when the user clicks the command, it is executed in terminal session 1. To select a different terminal session, you can set the `session` property.

```
```terminal:execute
command: echo "Execute command."
session: 1
```
```

To define a command the user clicks that executes in all terminal sessions on the terminals tab of the dashboard, you can also use:

```
```terminal:execute-all
command: echo "Execute command."
```
```

For `terminal:execute` or `terminal:execute-all`, to clear the terminal before the command is executed, set the `clear` property to `true`:

```
```terminal:execute
command: echo "Execute command."
clear: true
```
```

This clears the full terminal buffer and not just the displayed portion of the buffer.

With the new clickable actions, to indicate that a running command in a terminal session must be interrupted, use:

```
```terminal:interrupt
session: 1
```
```

(Optional) Set the `session` property within the code block to indicate an alternate terminal session to session 1.

To allow the user to send an interrupt to all terminals sessions on the terminals tab of the dashboard, use:

```
```terminal:interrupt-all
```
```

Where you want the user to enter input into a terminal rather than a command, such as when a running command prompts for a password, use:

```
```terminal:input
text: password
```
```

To allow the user to run commands or interrupt a command, set the `session` property to indicate a specific terminal to send it to if you don't want to send it to terminal session 1:

```
```terminal:input
text: password
session: 1
```
```

When providing terminal input in this way, the text by default still has a newline appended to the end, making it behave the same as using `terminal:execute`. If you do not want a newline appended, set the `endl` property to `false`.

```
```terminal:input
text: input
endl: false
```
```

To allow the user to clear all terminal sessions on the terminals tab of the dashboard, use:

```
```terminal:clear-all
```
```

This clears the full terminal buffer and not just the displayed portion of the terminal buffer. It does not have any effect when an application is running in the terminal using visual mode. To clear only the displayed portion of the terminal buffer when a command dialog box is displayed, use `terminal:execute` and run the `clear` command.

To allow the user to copy content to the paste buffer, use:

```
```workshop:copy
text: echo "Text to copy."
```
```

or:

```
```workshop:copy-and-edit
text: echo "Text to copy and edit."
```
```

A benefit of using these over the original methods is that by using the appropriate YAML syntax, you can control whether:

- A multiline string value is concatenated into one line.
- Line breaks are preserved.
- Initial or terminating new lines are included.

In the original methods, the string was always trimmed before use. By using the different forms as appropriate, you can annotate the displayed code block with a different message letting the user know what will happen.

The method for using AsciiDoc is similar, using the `role` for the name of the annotation and YAML as the content:

```
[source,bash,role=terminal:execute]
----
command: echo "Execute command."
----
```

Supported workshop editor

Learning Center currently **only** supports the code-server v4.4.0 of VS Code as an editor in workshops.

Clickable actions for the dashboard

In addition to the clickable actions related to the terminal and copying of text to the paste buffer, other actions are available for controlling the dashboard and opening URL links.

To allow the user to click in the workshop content to open a URL in a new browser, use:

```
```dashboard:open-url
url: https://www.example.com/
```
```

To allow the user to click in the workshop content to display a specific dashboard tab if hidden, use:

```
```dashboard:open-dashboard
name: Terminal
```
```

To allow the user to click in the workshop content to display the console tab, use:

```
```dashboard:open-dashboard
name: Console
```
```

To allow the user to click in the workshop content to display a specific view within the Kubernetes web console by using a clickable action block, rather than requiring the user to find the correct view, use:

```
```dashboard:reload-dashboard
name: Console
prefix: Console
title: List pods in namespace {{session_namespace}}
url: {{ingress_protocol}}://{{session_namespace}}-console.{{ingress_domain}}/#/pod?nam
espace={{session_namespace}}
description: ""
```
```

To allow the user to create a new dashboard tab with a specific URL, use:

```
```dashboard:create-dashboard
name: Example
url: https://www.example.com/
```
```

To allow the user to create a new dashboard tab with a new terminal session, use:

```dashboard:create-dashboard
name: Example
url: terminal:example
```

The value must be of the form `terminal:<session>`, where `<session>` is replaced with the name you want to give the terminal session. The terminal session name must be restricted to lowercase letters, numbers, and '-'. You must avoid using numeric terminal session names such as "1", "2", and "3", because these are used for the default terminal sessions.

To allow the user to reload an existing dashboard, using the URL it is currently targeting, use:

```dashboard:reload-dashboard
name: Example
```

If the dashboard is for a terminal session, there is no effect unless the terminal session was disconnected, in which case it is reconnected.

To allow the user to change the URL target of an existing dashboard by entering the new URL when reloading a dashboard, use:

```dashboard:reload-dashboard
name: Example
url: https://www.example.com/
```

The user cannot change the target of a dashboard that includes a terminal session.

To allow the user to delete a dashboard, use:

```dashboard:delete-dashboard
name: Example
```

The user cannot delete dashboards corresponding to builtin applications provided by the workshop environment, such as the default terminals, console, editor, or slides.

Deleting a custom dashboard including a terminal session does not destroy the underlying terminal session, and the user can reconnect it by creating a new custom dashboard for the same terminal session name.

Clickable actions for the editor

If the embedded editor is enabled, special actions are available that control the editor.

To allow the user to open an existing file you can use:

```editor:open-file
file: ~/exercises/sample.txt
```

You can use `~/` prefix to indicate the path relative to the home directory of the session. When the user opens the file, if you want the insertion point left on a specific line, provide the `line` property. Lines numbers start at `1`.

```editor:open-file
file: ~/exercises/sample.txt
line: 1
```

To allow the user to highlight certain lines of a file based on an exact string match, use:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "int main()"
```
```

The region of the match is highlighted by default. To allow the user to highlight any number of lines before or after the line with the match, you can set the `before` and `after` properties:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "int main()"
before: 1
after: 1
```
```

Setting both `before` and `after` to `0` causes the complete line that matched to be highlighted instead of a region within the line.

To match based on a regular expression, rather than an exact match, set `isRegex` to `true`:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "image: (.*)"
isRegex: true
```
```

When a regular expression is used, and subgroups are specified within the pattern, you can indicate which subgroup is selected:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "image: (.*)"
isRegex: true
group: 1
```
```

Where there are multiple possible matches in a file, and the one you want to match is not the first, you can set a range of lines to search:

```
```editor:select-matching-text
file: ~/exercises/sample.txt
text: "image: (.*)"
isRegex: true
start: 8
stop: 12
```
```

Absence of `start` means start at the beginning of the file. Absence of `stop` means stop at the end of the file. The line number given by `stop` is not included in the search.

For both an exact match and regular expression, the text to be matched must all be on one line. It is not possible to match text that spans across lines.

To allow the user to replace text within the file, first match it exactly or use a regular expression so it is marked as selected, then use:

```
```editor:replace-text-selection
file: ~/exercises/sample.txt
```
```

```
text: nginx:latest
```

To allow the user to append lines to the end of a file, use:

```
```editor:append-lines-to-file
file: ~/exercises/sample.txt
text: |
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
    do eiusmod tempor incididunt ut labore et dolore magna aliqua.
```
```

If the user runs the action `editor:append-lines-to-file` and the file doesn't exist, it is created. You can use this to create new files for the user.

To allow the user to insert lines before a specified line in the file, use:

```
```editor:insert-lines-before-line
file: ~/exercises/sample.txt
line: 8
text: |
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
    do eiusmod tempor incididunt ut labore et dolore magna aliqua.
```
```

To allow the user to insert lines after matching a line containing a specified string, use:

```
```editor:append-lines-after-match
file: ~/exercises/sample.txt
match: Lorem ipsum
text: |
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
    do eiusmod tempor incididunt ut labore et dolore magna aliqua.
```
```

Where the file contains YAML, to allow the user to insert a new YAML value into an existing structure, use:

```
```editor:insert-value-into-yaml
file: ~/exercises/deployment.yaml
path: spec.template.spec.containers
value:
- name: nginx
  image: nginx:latest
```
```

To allow the user to execute a registered VS code command, use:

```
```editor:execute-command
command: spring.initializr.maven-project
args:
- language: Java
  dependencies: [ "actuator", "webflux" ]
  artifactId: demo
  groupId: com.example
```
```

Clickable actions for file download

If file downloads are enabled for the workshop, you can use the `files:download-file` clickable action:

```
```files:download-file
path: .kube/config
```
```

The action triggers saving the file to the user's local computer, and the file is not displayed in the user's web browser.

Clickable actions for the examiner

If the test examiner is enabled, special actions are available to run verification checks to verify whether a workshop user has performed a required step. You can trigger these verification checks by clicking on the action, or you can configure them to start running when the page loads.

For a single verification check that the user must click to run, use:

```
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists.
args:
- one
```
```

The `title` field is displayed as the title of the clickable action and must describe the nature of the test. If required, you can provide a `description` field for a longer explanation of the test. This is displayed in the body of the clickable action but is shown as preformatted text.

There must be an executable program (script or compiled application) in the `workshop/examiner/tests` directory with name matching the value of the `name` field.

The list of program arguments against the `args` field is passed to the test program.

The executable program for the test must exit with a status of 0 if the test is successful and nonzero if the test is a failure. The test should aim to return as quickly as possible and should not be a persistent program.

```
#!/bin/bash

kubectl get pods --field-selector=status.phase=Running -o name | egrep -e "^pod/$1$"

if [ "$?" != "0" ]; then
    exit 1
fi

exit 0
```

By default, the program for a test is stopped after a timeout of 15 seconds, and the test is deemed to have failed. To adjust the timeout, you can set the `timeout` value, which is in seconds. A value of 0 causes the default 15 seconds timeout to be applied. It is not possible to deactivate stopping the test program after running for the default or a specified `timeout` value.

```
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
```
```

To apply the test multiple times, you can enable the retry when a failure occurs. For this you must set the number of times to retry and the delay between retries. The value for the delay is in

seconds.

```
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: 10
delay: 1
```
```

When you use retries, the testing stops as soon as the test program returns that it was successful.

To have retries continue for as long as the page of the workshop instructions displays, set `retries` to the special YAML value of `.INF`:

```
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: .INF
delay: 1
```
```

Rather than require a workshop user to click the action to run the test, you can have the test start as soon as the page is loaded, or when a section the page is contained in is expanded. Do this by setting `autostart` to `true`:

```
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: .INF
delay: 1
autostart: true
```
```

When a test succeeds, to immediately start the next test in the same page, set `cascade` to `true`.

```
```examiner:execute-test
name: test-that-pod-exists
title: Verify that pod named "one" exists
args:
- one
timeout: 5
retries: .INF
delay: 1
autostart: true
cascade: true
```

```examiner:execute-test
name: test-that-pod-does-not-exist
title: Verify that pod named "one" does not exist
args:
- one
retries: .INF
```
```

```
delay: 1
```

Clickable actions for sections

For optional instructions, or instructions you want to hide until the workshop user is ready for them, you can designate sections to be hidden. When the user clicks the appropriate action, the section expands to show its content. You can use this for examples that initially hide a set of questions or a test at the end of each workshop page.

In order to designate a section of content as hidden, you must use two separate action code blocks marking the beginning and end of the section:

````
```section:begin
title: Questions
```

To show you understand ...

```section:end
```
````

The `title` must be set to the text you want to include in the banner for the clickable action.

A clickable action is only shown for the beginning of the section, and the action for the end is always hidden. Clicking the action for the beginning expands the section. The user can collapse the section again by clicking the action.

To create nested sections, you must name the action blocks for the beginning and end so they are correctly matched:

````
```section:begin
name: questions
title: Questions
```

To show you understand ...

```section:begin
name: question-1
prefix: Question
title: 1
```

...

```section:end
name: question-1
```

```section:end
name: questions
```
````

The `prefix` attribute allows you to override the default `Section` prefix used on the title for the action.

If a collapsible section includes an examiner action block set to automatically run, it only starts when the user expands the collapsible section.

In case you want a section header showing in the same style as other clickable actions, you can use:

```
```section:heading
title: Questions
```
```

When the user clicks on this, the action is still marked as completed, but it does not trigger any other action.

Overriding title and description

Clickable action blocks default to use a title with the prefix dictated by what the action block does. The body of the action block also defaults to use a value commensurate with the action.

Especially for complicated scenarios involving editing of files, the defaults might not be the most appropriate and be confusing, so you can override them. To override these defaults, set the `prefix`, `title`, and `description` fields of a clickable action block:

```
```action:name
prefix: Prefix
title: Title
description: Description
```
```

The banner of the action block in this example displays “Prefix: Title”, with the body showing “Description”.

Note

The description is always displayed as pre-formatted text within the rendered page.

Escaping of code block content

Because the Liquid template engine is applied to workshop content, you must escape content in code blocks that conflict with the syntactic elements of the Liquid template engine. To escape such elements, you can suspend processing by the template engine for that section of workshop content to ensure it is rendered correctly. Do this by using a Liquid `{% raw %}...{% endraw %}` block.

```
{% raw %}
```execute
echo "Execute command."
```
{% endraw %}
```

This has the side effect of preventing interpolation of data variables, so restrict it to only the required scope.

Interpolation of data variables

When creating page content, you can reference a number of predefined data variables. The values of the data variables are substituted into the page when rendered in the user’s browser.

The workshop environment provides the following built-in data variables:

- `workshop_name`: The name of the workshop.
- `workshop_namespace`: The name of the namespace used for the workshop environment.

- `session_namespace`: The name of the namespace the workshop instance is linked to and into which any deployed applications run.
- `training_portal`: The name of the training portal the workshop is hosted by.
- `ingress_domain`: The host domain must be used in the any generated host name of ingress routes for exposing applications.
- `ingress_protocol`: The protocol (http/https) used for ingress routes created for workshops.

To use a data variable within the page content, surround it by matching pairs of brackets:

```
{{ session_namespace }}
```

Do this inside of code blocks, including clickable actions, as well as in URLs:

```
http://myapp-{{ session_namespace }}.{{ ingress_domain }}
```

When the workshop environment is hosted in Kubernetes and provides access to the underlying cluster, the following data variables are also available.

- `kubernetes_token`: The Kubernetes access token of the service account the workshop session is running as.
- `kubernetes_ca_crt`: The contents of the public certificate required when accessing the Kubernetes API URL.
- `kubernetes_api_url`: The URL for accessing the Kubernetes API. This is only valid when used from the workshop terminal.

Note

An older version of the rendering engine required that data variables be surrounded on each side with the character `%`. This is still supported for backwards compatibility, but VMware recommends you use matched pairs of brackets instead.

Adding custom data variables

You can introduce your own data variables by listing them in the `workshop/modules.yaml` file. A data variable is defined as having a default value, but the value is overridden if an environment variable of the same name is defined.

The field under which the data variables must be specified is `config.vars`:

```
config:
    vars:
    - name: LANGUAGE
      value: undefined
```

To use a name for a data variable that is different from the environment variable name, add a list of `aliases`:

```
config:
    vars:
    - name: LANGUAGE
      value: undefined
      aliases:
      - PROGRAMMING_LANGUAGE
```

The environment variables with names in the list of aliases are checked first, then the environment variable with the same name as the data variable. If no environment variables with those names are set, the default value is used.

You can override the default value for a data variable for a specific workshop by setting it in the corresponding workshop file. For example, `workshop/workshop-python.yaml` might contain:

```
vars:
  LANGUAGE: python
```

For more control over setting the values of data variables, you can provide the file `workshop/config.js`. The form of this file is:

```
function initialize(workshop) {
    workshop.load_workshop();

    if (process.env['WORKSHOP_FILE'] == 'workshop-python.yaml') {
        workshop.data_variable('LANGUAGE', 'python');
    }
}

exports.default = initialize;

module.exports = exports.default;
```

This JavaScript code is loaded and the `initialize()` function called to set up the workshop configuration. You can then use the `workshop.data_variable()` function to set up any data variables.

Because it is JavaScript, you can write any code to query process environment variables and set data variables based on those. This might include creating composite values constructed from multiple environment variables. You can even download data variables from a remote host.

Passing environment variables

You can pass environment variables, including remapping of variable names, by setting your own custom data variables. If you don't need to set default values or remap the name of an environment variable, you can instead reference the name of the environment variable directly. You must prefix the name with `ENV_` when using it.

For example, to display the value of the `KUBECTL_VERSION` environment variable in the workshop content, use `ENV_KUBECTL_VERSION`, as in:

```
{{ ENV_KUBECTL_VERSION }}
```

Handling embedded URL links

You can include URLs in workshop content. This can be the literal URL, or the Markdown or AsciiDoc syntax for including and labelling a URL. What happens when a user clicks on a URL depends on the specific URL.

In the case of the URL being an external website, when the URL is clicked, the URL opens in a new browser tab or window. When the URL is a relative page referring to another page that is part of the workshop content, the page replaces the current workshop page.

You can define a URL where components of the URL are provided by data variables. Data variables useful for this are `session_namespace` and `ingress_domain`, because they can be used to create a URL to an application deployed from a workshop:

```
https://myapp-{{ session_namespace }}.{{ ingress_domain }}
```

Conditional rendering of content

Rendering pages is in part handled by the Liquid template engine. So you can use any constructs the template engine supports for conditional content:

```
{% if LANGUAGE == 'java' %}
....
{% endif %}
{% if LANGUAGE == 'python' %}
....
{% endif %}
```

Embedding custom HTML content

Custom HTML can be embedded in the workshop content by using the appropriate mechanism provided by the content rendering engine used.

If using Markdown, HTML can be embedded directly without being marked as HTML:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin justo.

<div>
<table style="width:100%">
  <thead>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  </thead>
  <tbody>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
  </tbody>
</table>
</div>

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin justo.
```

If using AsciiDoc, HTML can be embedded by using a passthrough block:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin justo.

++++
<div>
<table style="width:100%">
  <thead>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
```

```
  </tr>
  </thead>
  <tbody>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
  </tbody>
</table>
</div>
++++

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin justo.
```

In both cases, VMware recommends that the HTML consist of only a single HTML element. If you have more than one, include them all in a `div` element. The latter is necessary if any of the HTML elements are marked as hidden and the embedded HTML is a part of a collapsible section. If you don't ensure the hidden HTML element is placed under the single top-level `div` element, the hidden HTML element is visible when the collapsible section is expanded.

In addition to visual HTML elements, you can also include elements for embedded scripts or style sheets.

If you have HTML markup that must be added to multiple pages, extract it into a separate file and use the include file mechanism of the Liquid template engine. You can also use the partial render mechanism of Liquid as a macro mechanism for expanding HTML content with supplied values.

Automate your Learning Center workshop runtime

Your workshop content can script the steps a user must run for a workshop. This topic tells you how to set this up.

In some cases, you must parameterize that content with information from the runtime environment. Data variables in workshop content allow this to a degree, but you can automate this by using scripts executed in the workshop container to set up configuration files.

Do this by supplying setup scripts that run when the container is started. You can also run persistent background processes in the container that perform extra work for you while a workshop is being run.

Predefined environment variables

When you create the workshop content, you can use data variables to automatically insert values corresponding to the specific workshop session or environment. For example: the name of the namespace used for the session and the ingress domain when creating an ingress route.

These data variables can display a YAML/JSON resource file in the workshop content with values already filled out. You can have executable commands that have the data variables substituted with values given as arguments to the commands.

For commands run in the shell environment, you can also reference a number of predefined environment variables.

Key environment variables are:

- `WORKSHOP_NAMESPACE` - The name of the namespace used for the workshop environment.

- `SESSION_NAMESPACE` - The name of the namespace the workshop instance is linked to and into which any deployed applications run.
- `INGRESS_DOMAIN` - The host domain that must be used in any generated host name of ingress routes for exposing applications.
- `INGRESS_PROTOCOL` - The protocol (http/https) used for ingress routes created for workshops.

Instead of having an executable command in the workshop content, use:

```
```execute
kubectl get all -n %session_namespace%
```
```

With the value of the session namespace filled out when the page is rendered, you can use:

```
```execute
kubectl get all -n $SESSION_NAMESPACE
```
```

The shell inserts the value of the environment variable.

Running steps on container start

To run a script that makes use of the earlier environment variables when the container is started, and to perform tasks such as pre-create YAML/JSON resource definitions with values filled out, you can add an executable shell script to the `workshop/setup.d` directory. The name of the executable shell script must have a `.sh` suffix to be recognized and run.

If the container is restarted, the setup script runs again in the new container. If the shell script is performing actions against the Kubernetes REST API using kubectl or by using another means, the actions it performs must be tolerant of running more than once.

When using a setup script to fill out values in resource files, a useful utility is `envsubst`. You can use this in a setup script as follows:

```
#!/bin/bash

envsubst < frontend/ingress.yaml.in > frontend/ingress.yaml
```

A reference of the form `${INGRESS_DOMAIN}` in the input file is replaced with the value of the `INGRESS_DOMAIN` environment variable.

Setup scripts have the `/home/eduk8s` directory as the current working directory.

If you are creating or updating files in the file system and using a custom workshop image, ensure that the workshop image is created with correct file permissions to allow updates.

Running background applications

The setup scripts run once on container startup. You can use the script to start a background application needed to run in the container for the life of the workshop, but if that application stops, it does not restart.

If you must run a background application, you can integrate the management of the background application with the supervisor daemon run within the container. To have the supervisor daemon manage the application for you, add a configuration file snippet for the supervisor daemon in the `workshop/supervisor` directory. This configuration file must have a `.conf` extension.

The form of the configuration file snippet must be:

```
[program:myapplication]
process_name=myapplication
command=/opt/myapplication/sbin/start-myapplication
stdout_logfile=/proc/1/fd/1
stdout_logfile_maxbytes=0
redirect_stderr=true
```

The application must send any logging output to `stdout` or `stderr`, and the configuration snippet must direct log output to `/proc/1/fd/1` so it is captured in the container log file. If you must restart or shut down the application within the workshop interactive terminal, you can use the `supervisorctl` control script.

Terminal user shell environment

Neither the setup scripts that run when the container starts nor background applications affect the user environment of the terminal shell. The shell environment makes use of `bash` and the `$HOME/.bash_profile` script is read to perform added setup for the user environment. Because some default setup is included in `$HOME/.bash_profile`, you must not replace it, because you can loose that configuration.

To provide commands to initialize each shell environment, you can provide the file `workshop/profile`. When this file exists, it is sourced at the end of the `$HOME/.bash_profile` file when it is processed.

Overriding terminal shell command

The user starts each terminal session by using the `bash` terminal shell. A terminal prompt dialog box displays, allowing the user to manually enter commands or perform clickable actions targetting the terminal session.

To specify the command to run for a terminal session, you can supply an executable shell script file in the `workshop/terminal` directory.

The name of the shell script file for a terminal session must be of the form `<session>.sh`, where `<session>` is replaced with the name of the terminal session. The session names of the default terminals configured to be displayed with the dashboard are `1`, `2`, and `3`.

The shell script file might be used to run a terminal-based application such as `k9s`, or to create an SSH session to a remote system.

```
#!/bin/bash

exec k9s
```

If the command that is run exits, the terminal session is marked as exited and you need to reload that terminal session to start over again. Alternatively, you could write the shell script file as a loop so it restarts the command you want to run if it ever exits.

```
#!/bin/bash

while true; do
    k9s
    sleep 1
done
```

If you want to run an interactive shell and output a banner at the start of the session with special information for the user, use a script file to output the banner and then run the interactive shell:

```
#!/bin/bash

echo
echo "Your session namespace is "$SESSION_NAMESPACE".
echo

exec bash
```

Add presenter slides to your Learning Center workshop

If your workshop includes a presentation, include slides by placing them in the `workshop/slides` directory. Anything in this directory is served up as static files through a HTTP web server. The default webpage must be provided as `index.html`.

Use reveal.js presentation tool

To support the use of reveal.js, static media assets for that package are already bundled and available at the standard URL paths that the package expects. You can drop your slide presentation using reveal.js into the `workshop/slides` directory and it will work with no additional setup.

If you are using reveal.js for the slides and you have history enabled or are using section IDs to support named links, you can use an anchor to a specific slide and that slide will be opened when clicked on:

```
%slides_url%#/questions
```

When using embedded links to the slides in workshop content, if the workshop content is displayed as part of the dashboard, the slides open in the tab to the right rather than as a separate browser window or tab.

Use a PDF file for presenter slides

For slides bundled as a PDF file, add the PDF file to `workshop/slides` and then add an `index.html` which displays the PDF embedded in the page.

Requirements for Learning Center in an air-gapped environment

This topic gives you the list of configurations required for Learning Center to properly function in an air-gapped environment.

Learning Center can run in an air-gapped environment but workshops do not have this capability by default. Users must therefore take the following steps to ensure Learning Center functions as expected.

Workshop yaml changes

In an air-gapped environment a user has no Internet access, so workshop yamls should be modified to use:

1. Private container registries.
2. Private Maven, NPM, Python, Go, or any other language repository.

For example, in NPM you can modify the npmrc file to use:

```
// .npmrc
registry=https://myregistry-url
```

Self-signed certificates

Air-gapped environments normally use private Certificate Authorities (CA) that may require the use of self-signed certificates. You can allow the injection of CAs by:

1. Setting the env variable NODE_EXTRA_CA_CERTS to the path of the file that contains one or more trusted certificates in PEM format.
2. Add the following to your workshop definition:

```
spec:
  session:
    env:
    - name: NODE_EXTRA_CA_CERTS
      value: "$my-cert-pathway"
```

Internet dependencies

If the workshop requires the installation of any Internet dependency, such as a Linux Tool or any other tool, it must be done in the workshop image. See Build an image

Define custom resources for Learning Center

This topic describes how you define custom resources for Learning Center workshops and training portals.

You can deploy workshop images directly to a container runtime. The Learning Center Operator enables managing the deployments into a Kubernetes cluster. A set of Kubernetes custom resource definitions (CRDs) controls the operation of the Learning Center Operator.

Note

The examples do not show all the possible fields of each custom resource type. Later documentation may go in-depth on possible fields and their definitions.

Workshop definition resource

The `Workshop` custom resource defines a workshop. It specifies the title and description of the workshop, the location of the workshop content or container image that you deploy, any resources that you pre-create in the workshop environment or for each instance of the workshop.

You can also define environment variables for the workshop image, the amount of CPU and memory resources for the workshop instance, any overall quota you will apply to the created namespaces and what the workshop uses.

A minimal example of the `Workshop` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
```

```
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

When you create an instance of the `Workshop` custom resource, the Learning Center Operator does not take any immediate action. This custom resource exists only to define the workshop.

You create the `Workshop` custom resource at the cluster scope.

Workshop environment resource

You must create a workshop environment first to deploy the instances of a workshop. The `WorkshopEnvironment` custom resource defines the configuration of the workshop environment and the details of the workshop that you deploy.

A minimal example of the `WorkshopEnvironment` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    token: lab-markdown-sample
  session:
    username: learningcenter
```

When you create an instance of the `WorkshopEnvironment` custom resource, the Learning Center Operator responds by creating a namespace to host the workshop instances. The `Workshop` resource defines the workshop instance and the `spec.workshop.name` field specifies the name of the `Workshop` resource. The namespace you create uses the same name as that of the `metadata.name` field in the `WorkshopEnvironment` resource.

The `spec.request.token` field defines a token with which you must supply a request to create an instance of a workshop in this workshop environment. If necessary, you can also specify the namespaces from which a request for a workshop instance to initiate.

The `Workshop` defines a set of common resources that must exist for the workshop. Learning Center Operator creates these common resources after you created the namespace for the workshop environment. If necessary, these resources can include creation of separate namespaces with specific resources that you create in those namespaces instead.

You create the `WorkshopEnvironment` custom resource at the cluster scope.

Workshop request resource

To create an instance of the workshop under the workshop environment, the typical path is to create an instance of the `WorkshopRequest` custom resource.

The `WorkshopRequest` custom resource is namespaced to allow who can create it. Role-based access control (RBAC) controls the request to create a workshop instance. This means you can

allow non-privileged users to create workshops, although the deployment of the workshop instance might require elevated privileges.

A minimal example of the `WorkshopRequest` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
    token: lab-markdown-sample
```

Apart from appropriate access from RBAC, the user requesting a workshop instance must know the name of the workshop environment and the secret token that permits workshop requests against that specific workshop environment.

You do not need to create the `WorkshopRequest` resource when you use the `TrainingPortal` resource to provide a web interface for accessing workshops. You only need to create the `WorkshopRequest` resource when you create the `WorkshopEnvironment` resource manually and do not use the training portal.

Workshop session resource

Although `WorkshopRequest` is the typical way to request workshop instances, the Learning Center Operator itself creates an instance of a `WorkshopSession` custom resource when the request is granted.

The `WorkshopSession` custom resource is the expanded definition of what the workshop instance is. It combines details from `Workshop` and `WorkshopEnvironment`, and also links back to the `WorkshopRequest` resource object that triggered the request. The Learning Center Operator reacts to an instance of `WorkshopSession` and creates the workshop instance based on that definition.

You create the `WorkshopSession` custom resource at the cluster scope.

Training portal resource

The `TrainingPortal` custom resource provides a high-level mechanism for creating a set of workshop environments and populating them with workshop instances.

A minimal example of the `TrainingPortal` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 1
```

You can set the capacity of the training room, which dictates how many workshop instances are created for each workshop.

You create the `TrainingPortal` custom resource at the cluster scope.

System profile resource

The `SystemProfile` custom resource provides a mechanism for configuring the Learning Center Operator. This provides additional features that use environment variables to configure the operator.

A minimal example of the `SystemProfile` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  ingress:
    domain: learningcenter.tanzu.vmware.com
    secret: learningcenter-tanzu-vmware-com-tls
    class: nginx
  environment:
    secrets:
      pull:
      - cluster-image-registry-pull
```

The operator, by default, looks for a default system profile called `default-system-profile`. Setting the `SYSTEM_PROFILE` environment variable on the deployment for the operator or using the `system.profile` setting on `TrainingPortal`, `WorkshopEnvironment`, or `WorkshopSession` custom resources for specific deployments can override the default name globally.

As only a global deployment of the operator is supported, the `SystemProfile` custom resource is created at cluster scope.

You can make changes to instances of the `SystemProfile` custom resource. The Learning Center Operator uses these changes without needing to redeploy the custom resource.

You create the `SystemProfile` custom resource at the cluster scope.

Loading the workshop CRDs

The custom resource definitions for the custom resource described earlier are created in the Kubernetes cluster when you deploy the Learning Center operator by using the Tanzu CLI.

This is because `v1` versions of CRDs are only supported from Kubernetes v1.17. If you want to use the `v1` versions of the CRDs, you must create a copy of the Learning Center operator deployment resources and override the configuration.

Define custom resources for Learning Center

This topic describes how you define custom resources for Learning Center workshops and training portals.

You can deploy workshop images directly to a container runtime. The Learning Center Operator enables managing the deployments into a Kubernetes cluster. A set of Kubernetes custom resource definitions (CRDs) controls the operation of the Learning Center Operator.

Note

The examples do not show all the possible fields of each custom resource type. Later documentation may go in-depth on possible fields and their definitions.

Workshop definition resource

The `Workshop` custom resource defines a workshop. It specifies the title and description of the workshop, the location of the workshop content or container image that you deploy, any resources that you pre-create in the workshop environment or for each instance of the workshop.

You can also define environment variables for the workshop image, the amount of CPU and memory resources for the workshop instance, any overall quota you will apply to the created namespaces and what the workshop uses.

A minimal example of the `Workshop` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
  session:
    namespaces:
      budget: small
    applications:
      console:
        enabled: true
      editor:
        enabled: true
```

When you create an instance of the `Workshop` custom resource, the Learning Center Operator does not take any immediate action. This custom resource exists only to define the workshop.

You create the `Workshop` custom resource at the cluster scope.

Workshop environment resource

You must create a workshop environment first to deploy the instances of a workshop. The `WorkshopEnvironment` custom resource defines the configuration of the workshop environment and the details of the workshop that you deploy.

A minimal example of the `WorkshopEnvironment` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    token: lab-markdown-sample
  session:
    username: learningcenter
```

When you create an instance of the `WorkshopEnvironment` custom resource, the Learning Center Operator responds by creating a namespace to host the workshop instances. The `Workshop` resource defines the workshop instance and the `spec.workshop.name` field specifies the name of the `Workshop` resource. The namespace you create uses the same name as that of the `metadata.name` field in the `WorkshopEnvironment` resource.

The `spec.request.token` field defines a token with which you must supply a request to create an instance of a workshop in this workshop environment. If necessary, you can also specify the namespaces from which a request for a workshop instance to initiate.

The `Workshop` defines a set of common resources that must exist for the workshop. Learning Center Operator creates these common resources after you created the namespace for the workshop environment. If necessary, these resources can include creation of separate namespaces with specific resources that you create in those namespaces instead.

You create the `WorkshopEnvironment` custom resource at the cluster scope.

Workshop request resource

To create an instance of the workshop under the workshop environment, the typical path is to create an instance of the `WorkshopRequest` custom resource.

The `WorkshopRequest` custom resource is namespaced to allow who can create it. Role-based access control (RBAC) controls the request to create a workshop instance. This means you can allow non-privileged users to create workshops, although the deployment of the workshop instance might require elevated privileges.

A minimal example of the `WorkshopRequest` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
    token: lab-markdown-sample
```

Apart from appropriate access from RBAC, the user requesting a workshop instance must know the name of the workshop environment and the secret token that permits workshop requests against that specific workshop environment.

You do not need to create the `WorkshopRequest` resource when you use the `TrainingPortal` resource to provide a web interface for accessing workshops. You only need to create the `WorkshopRequest` resource when you create the `WorkshopEnvironment` resource manually and do not use the training portal.

Workshop session resource

Although `WorkshopRequest` is the typical way to request workshop instances, the Learning Center Operator itself creates an instance of a `WorkshopSession` custom resource when the request is granted.

The `WorkshopSession` custom resource is the expanded definition of what the workshop instance is. It combines details from `Workshop` and `WorkshopEnvironment`, and also links back to the `WorkshopRequest` resource object that triggered the request. The Learning Center Operator reacts to an instance of `WorkshopSession` and creates the workshop instance based on that definition.

You create the `WorkshopSession` custom resource at the cluster scope.

Training portal resource

The `TrainingPortal` custom resource provides a high-level mechanism for creating a set of workshop environments and populating them with workshop instances.

A minimal example of the `TrainingPortal` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
```

```
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 1
```

You can set the capacity of the training room, which dictates how many workshop instances are created for each workshop.

You create the `TrainingPortal` custom resource at the cluster scope.

System profile resource

The `SystemProfile` custom resource provides a mechanism for configuring the Learning Center Operator. This provides additional features that use environment variables to configure the operator.

A minimal example of the `SystemProfile` custom resource looks like this:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  ingress:
    domain: learningcenter.tanzu.vmware.com
    secret: learningcenter-tanzu-vmware-com-tls
    class: nginx
  environment:
    secrets:
      pull:
      - cluster-image-registry-pull
```

The operator, by default, looks for a default system profile called `default-system-profile`. Setting the `SYSTEM_PROFILE` environment variable on the deployment for the operator or using the `system.profile` setting on `TrainingPortal`, `WorkshopEnvironment`, or `WorkshopSession` custom resources for specific deployments can override the default name globally.

As only a global deployment of the operator is supported, the `SystemProfile` custom resource is created at cluster scope.

You can make changes to instances of the `SystemProfile` custom resource. The Learning Center Operator uses these changes without needing to redeploy the custom resource.

You create the `SystemProfile` custom resource at the cluster scope.

Loading the workshop CRDs

The custom resource definitions for the custom resource described earlier are created in the Kubernetes cluster when you deploy the Learning Center operator by using the Tanzu CLI.

This is because `v1` versions of CRDs are only supported from Kubernetes v1.17. If you want to use the `v1` versions of the CRDs, you must create a copy of the Learning Center operator deployment resources and override the configuration.

Configure the Workshop resource

This topic describes how you configure the `Workshop` custom resource, which defines a Learning Center workshop.

Workshop title and description

Each workshop must have the `title` and `description` fields. If you do not supply these fields, the `Workshop` resource is rejected when you attempt to load it into the Kubernetes cluster.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
```

Where:

- The `title` field has a single-line value specifying the subject of the workshop.
- The `description` field has a longer description of the workshop.

You can also supply the following optional information for the workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  url: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
  difficulty: beginner
  duration: 15m
  vendor: learningcenter.tanzu.vmware.com
  authors:
  - John Smith
  tags:
  - template
  logo: ....
  content:
    files: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where:

- The `url` field is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`. It must be a URL you can use to get more information about the workshop.
- The `difficulty` field indicates the target audiences of the workshop. The value can be `beginner`, `intermediate`, `advanced`, or `extreme`.
- The `duration` field gives the maximum amount of time the workshop takes to complete. This field provides informational value and does not guarantee how long a workshop instance lasts. The field format is an integer number with `s`, `m`, or `h` suffix.
- The `vendor` field must be a value that identifies the company or organization with which the authors are affiliated. This is a company or organization name or a DNS host name under the control of whoever has created the workshop.
- The `authors` field must list the people who create the workshop.
- The `tags` field must list labels identifying what the workshop is about. This is used in a searchable catalog of workshops.

- The `logo` field must be an image provided in embedded data URI format that depicts the topic of the workshop. The image must be 400 by 400 pixels. You can use it in a searchable catalog of workshops.
- The `files` field is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

When referring to a workshop definition after you load it into a Kubernetes cluster, use the value of the `name` field given in the metadata. To experiment with different variations of a workshop, copy the original workshop definition YAML file and change the value of `name`. Make your changes and load it into the Kubernetes cluster.

Downloading workshop content

You can download workshop content when you create the workshop instance. If the amount of content is moderate, the download doesn't increase startup time for the workshop instance. The alternative is to bundle the workshop content in a container image you build from the Learning Center workshop base image.

To download workshop content at the time the workshop instance starts, set the `content.files` field to the location of the workshop content:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
```

The location is a GitHub or GitLab repository, a URL to a tarball hosted on a HTTP server, or a reference to an OCI image artifact on a registry.

For a GitHub or GitLab repository, do not prefix the location with `https://` as it uses symbolic reference and is not a URL.

The format of the reference to a GitHub or GitLab repository is similar to what you use with Kustomize when referencing remote repositories. For example:

- `github.com/organisation/project?ref=develop` or `github.com/organisation/project?ref=main`: Use the workshop content you host at the root of the GitHub repository. Use the `develop` or `main` branch. Be sure to specify the ref branch, because not specifying the branch may lead to content download errors.
- `github.com/organisation/project/subdir?ref=develop`: Use the workshop content you host at `subdir` of the GitHub repository. Use the `develop` branch.
- `gitlab.com/organisation/project`: Use the workshop content you host at the root of the GitLab repository. Use the `main` branch.
- `gitlab.com/organisation/project/subdir?ref=develop`: Use the workshop content you host at `subdir` of the GitLab repository. Use the `develop` branch.

For a URL to a tarball hosted on a HTTP server, the URL is in the following formats:

- `https://example.com/workshop.tar` - Use the workshop content from the top-level directory of the unpacked tarball.
- `https://example.com/workshop.tar.gz` - Use the workshop content from the top-level directory of the unpacked tarball.

- `https://example.com/workshop.tar?path=subdir` - Use the workshop content from the subdirectory path of the unpacked tarball.
- `https://example.com/workshop.tar.gz?path=subdir` - Use the workshop content from the subdirectory path of the unpacked tarball.

The tarball referenced by the URL is either uncompressed or compressed.

For GitHub, instead of referencing the Git repository containing the workshop content, use a URL to refer directly to the downloadable tarball for a specific version of the Git repository:

- `https://github.com/organization/project/archive/develop.tar.gz?path=project-develop`

You must reference the `.tar.gz` download and cannot use the `.zip` file. The base name of the tarball file is the branch or commit name. You must enter the `path` query string parameter where the argument is the name of the project and branch or project and commit. You must supply the path because the contents of the repository are not returned at the root of the archive.

GitLab also provides a means of downloading a package as a tarball:

- `https://gitlab.com/organization/project/-/archive/develop/project-develop.tar.gz?path=project-develop`

If the GitHub or GitLab repository is private, you can generate a personal access token providing read-only access to the repository and include the credentials in the URL:

- `https://username@token:github.com/organization/project/archive/develop.tar.gz?path=project-develop`

With this method, you supply a full URL to request a tarball of the repository and it does not refer to the repository itself. You can also reference private enterprise versions of GitHub or GitLab and the repository doesn't need to be on the public `github.com` or `gitlab.com` sites.

The last case is a reference to an OCI image artifact stored on a registry. This is not a full container image with the operating system, but an image containing only the files making up the workshop content. The URI formats for this are:

- `imgpkg+https://harbor.example.com/organisation/project:version` - Use the workshop content from the top-level directory of the unpacked OCI artifact. The registry in this case must support `https`.
- `imgpkg+https://harbor.example.com/organisation/project:version?path=subdir` - Use the workshop content from the subdirectory path of the unpacked OCI artifact you specify. The registry in this case must support `https`.
- `imgpkg+http://harbor.example.com/organisation/project:version` - Use the workshop content from the top-level directory of the unpacked OCI artifact. The registry in this case can only support `http`.
- `imgpkg+http://harbor.example.com/organisation/project:version?path=subdir` - Use the workshop content from the subdirectory path of the unpacked OCI artifact you specify. The registry in this case can only support `http`.

You can use `imgpkg://` instead of the prefix `imgpkg+https://`. The registry in this case must still support `https`.

For any of the formats, you can supply credentials as part of the URI:

- `imgpkg+https://username:password@harbor.example.com/organisation/project:version`

Access to the registry using a secure connection of `https` must have a valid certificate.

You can create the OCI image artifact by using `imgpkg` from the Carvel tool set. For example, from the top-level directory of the Git repository containing the workshop content, run:

```
imgpkg push -i harbor.example.com/organisation/project:version -f .
```

In all cases for downloading workshop content, the `workshop` subdirectory holding the actual workshop content is relocated to `/opt/workshop` so that it is not visible to a user. If you want to ignore other files so the user can not see them, you can supply a `.eduk8signore` file in your repository or tarball and list patterns for the files in it.

The contents of the `.eduk8signore` file are processed as a list of patterns and each is applied recursively to subdirectories. To ensure that a file is only ignored if it resides in the root directory, prefix it with `./`:

```
./.dockerignore
./.gitignore
./Dockerfile
./LICENSE
./README.md
./kustomization.yaml
./resources
```

Container image for the workshop

When you bundle the workshop content into a container image, the `content.image` field must specify the image reference identifying the location of the container image that you will deploy for the workshop instance:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
```

Even though you can download workshop content when the workshop environment starts, you might still want to override the workshop image that is used as a base. You can do this when you have a custom workshop base image that includes added language runtimes or tools that the specialized workshops require.

For example, if running a Java workshop, you can enter the `jdk11-environment` for the workshop image. The workshop content is still downloaded from GitHub:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-spring-testing
spec:
  title: Spring Testing
  description: Playground for testing Spring development
  content:
    image: registry.tanzu.vmware.com/learning-center/jdk11-environment:latest
    files: {YOUR-GIT-REPO-URL}/lab-spring-testing
```

If you want to use the latest version of an image, always include the `:latest` tag. This is important because the Learning Center Operator looks for version tags `:main`, `:main`, `:develop` and `:latest`. When using these tags, the Operator sets the image pull policy to `Always` to ensure that a newer version is always pulled if available. Otherwise, the image is cached on the Kubernetes nodes and only pulled when it is initially absent. Any other version tags are always assumed to be unique and

are never updated. Be aware of image registries that use a content delivery network (CDN) as front end. When using these image tags, the CDN can still regard them as unique and not do pull through requests to update an image even if it uses a tag of `:latest`.

When special custom workshop base images are available as part of the Learning Center project, instead of specifying the full location for the image, including the image registry, you can specify a short name. The Learning Center Operator then fills in the rest of the details:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-spring-testing
spec:
  title: Spring Testing
  description: Playground for testing Spring development
  content:
    image: jdk11-environment:latest
    files: github.com/eduk8s-tests/lab-spring-testing
```

The supported short versions of the names are:

- `base-environment:*`: A tagged version of the `base-environment` workshop image matched with the current version of the Learning Center Operator.

The `*` variants of the short names map to the most up-to-date version of the image available when the version of the Learning Center Operator was released. That version is guaranteed to work with that version of the Learning Center Operator. The `latest` version can be newer, with possible incompatibilities.

If required, you can remap the short names in the `SystemProfile` configuration of the Learning Center Operator. You can map additional short names to your own custom workshop base images for your own deployment of the Learning Center Operator, and with any of your own workshops.

Setting environment variables

To set or override environment variables for the workshop instance, you can supply the `session.env` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    files: {YOUR-GIT-REPO-URL}/lab-markdown-sample
  session:
    env:
    - name: REPOSITORY-URL
      value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where:

- The `session.env` field is a list of dictionaries with the `name` and `value` fields.
- The `value` field is the Git repository for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `session_id`: A unique ID for the workshop instance within the workshop environment.

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session. A workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances. It is also the namespace where the service account that the workshop instance runs.
- `service_account`: The name of the service account that the workshop instance runs as. It has access to the namespace you create for that workshop instance.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) you use for ingress routes and create for workshops.

The syntax for referencing the parameters is `$(parameter_name)`.

Use the `session.env` field to override environment variables only when they are required for the workshop. To set or override an environment for a specific workshop environment, set environment variables in the `WorkshopEnvironment` custom resource for the workshop environment instead.

Overriding the memory available

By default the container the workshop environment runs in is allocated 512Mi. If the editor is enabled, a total of 1Gi is allocated.

The memory allocation is sufficient for the workshop that is mainly aimed at deploying workloads into the Kubernetes cluster. If you run workloads in the workshop environment container and need more memory, you can override the default by setting `memory` under `session.resources`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    resources:
      memory: 2Gi
```

Mounting a persistent volume

In circumstances where a workshop needs persistent storage to ensure no loss of work, you can request a persistent volume be mounted into the workshop container after the workshop environment container is stopped and restarted:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
```

```
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    resources:
      storage: 5Gi
```

The persistent volume is mounted on top of the `/home/eduk8s` directory. Because this hides any workshop content bundled with the image, an init container is automatically configured and run, which copies the contents of the home directory to the persistent volume before the persistent volume is mounted on top of the home directory.

Resource budget for namespaces

In conjunction with each workshop instance, a namespace is created during the workshop. From the terminal of the workshop, you can deploy dashboard applications into the namespace through the Kubernetes REST API by using tools such as kubectl.

By default, this namespace has all the limit ranges and resource quotas the Kubernetes cluster can enforce. In most cases, this means there are no limits or quotas.

To control how much resources you can use when you set no limit ranges and resource quotas, or override any default limit ranges and resource quotas, you can set a resource budget for any namespace of the workshop instance in the `session.namespaces.budget` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    namespaces:
      budget: small
```

The resource budget sizings and quotas for CPU and memory are:

Budget	CPU	Memory
small	1000m	1Gi
medium	2000m	2Gi
large	4000m	4Gi
x-large	8000m	8Gi
xx-large	8000m	12Gi
xxx-large	8000m	16Gi

A value of 1000m is equivalent to 1 CPU.

Separate resource quotas for CPU and memory are applied for terminating and non-terminating workloads.

Only the CPU and memory quotas are listed in the preceding table, but limits also apply to the number of resource objects of certain types you can create, such as:

- persistent volume claims
- replication controllers

- services
- secrets

For each budget type, a limit range is created with fixed defaults. The limit ranges for CPU usage on a container are as follows:

Budget	Minimum	Maximum	Request	Limit
small	50m	1000m	50m	250m
medium	50m	2000m	50m	500m
large	50m	4000m	50m	500m
x-large	50m	8000m	50m	500m
xx-large	50m	8000m	50m	500m
xxx-large	50m	8000m	50m	500m

The limit ranges for memory are as follows:

Budget	Minimum	Maximum	Request	Limit
small	32Mi	1Gi	128Mi	256Mi
medium	32Mi	2Gi	128Mi	512Mi
large	32Mi	4Gi	128Mi	1Gi
x-large	32Mi	8Gi	128Mi	2Gi
xx-large	32Mi	12Gi	128Mi	2Gi
xxx-large	32Mi	16Gi	128Mi	2Gi

The request and limit values are the defaults of a container when there is no resources specification in a pod specification.

You can supply overrides in `session.namespaces.limits` to override the limit ranges and defaults for request and limit values when a budget sizing for CPU and memory is sufficient and there is no resources specification in a pod specification:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-markdown-sample
spec:
  title: Markdown Sample
  description: A sample workshop using Markdown
  content:
    image: {YOUR-REGISTRY-URL}/lab-markdown-sample:main
  session:
    namespaces:
      budget: medium
      limits:
        min:
          cpu: 50m
          memory: 32Mi
        max:
          cpu: 1
          memory: 1Gi
        defaultRequest:
          cpu: 50m
          memory: 128Mi
        default:
```

```
          cpu: 500m
          memory: 1Gi
```

Although all the configurable properties are listed in this example, you only need to supply the property for the value that you want to override.

If you need more control over the limit ranges and resource quotas, you can set the resource budget to `custom`. This removes any default limit ranges and resource quota that might be applied to the namespace. You can enter your own `LimitRange` and `ResourceQuota` resources as part of the list of resources created for each session.

Before disabling the quota and limit ranges or contemplating any switch to using a custom set of `LimitRange` and `ResourceQuota` resources, consider if that is what is really required.

The default requests defined by these for memory and CPU are fallbacks only. In most cases, instead of changing the defaults, you can enter the memory and CPU resources in the pod template specification of your deployment resources used in the workshop to indicate what the application requires. This allows you to control exactly what the application can use and so fit into the minimum quota required for the task.

This budget setting and the memory values are distinct from the amount of memory the container the workshop environment runs in. To change how much memory is available to the workshop container, set the `memory` setting under `session.resources`.

Patching workshop deployment

In order to set or override environment variables, you can provide `session.env`. To make other changes to the Pod template for the deployment used to create the workshop instance, provide an overlay patch. You can use this patch to override the default CPU and memory limit applied to the workshop instance or to mount a volume.

The patches are provided by setting `session.patches`. The patch is applied to the `spec` field of the pod template:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-resource-testing
spec:
  title: Resource testing
  description: Play area for testing memory resources
  content:
    files: github.com/eduk8s-tests/lab-resource-testing
  session:
    patches:
      containers:
      - name: workshop
        resources:
          requests:
            memory: "1Gi"
          limits:
            memory: "1Gi"
```

In this example, the default memory limit of “512Mi” is increased to “1Gi”. Although memory is set using a patch in this example, the `session.resources.memory` field is the preferred way to override the memory allocated to the container the workshop environment is running in.

The patch works differently than overlay patches that you can find elsewhere in Kubernetes. Specifically, when patching an array and the array contains a list of objects, a search is performed on the destination array. If an object already exists with the same value for the `name` field, the item in the source array is overlaid on top of the existing item in the destination array.

If there is no matching item in the destination array, the item in the source array is added to the end of the destination array.

This means an array doesn't outright replace an existing array, but a more intelligent merge is performed of elements in the array.

Creation of session resources

When a workshop instance is created, the deployment running the workshop dashboard is created in the namespace for the workshop environment. When more than one workshop instance is created under that workshop environment, all those deployments are in the same namespace.

For each workshop instance, a separate empty namespace is created with name corresponding to the workshop session. The workshop instance is configured so that the service account that the workshop instance runs under can access and create resources in the namespace created for that workshop instance. Each separate workshop instance has its own corresponding namespace and cannot see the namespace for another instance.

To pre-create additional resources within the namespace for a workshop instance, you can supply a list of the resources against the `session.objects` field within the workshop definition. You might use this to add additional custom roles to the service account for the workshop instance when working in that namespace or to deploy a distinct instance of an application for just that workshop instance, such as a private image registry:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-registry-testing
spec:
  title: Registry Testing
  description: Play area for testing image registry
  content:
    files: github.com/eduk8s-tests/lab-registry-testing
  session:
    objects:
    - apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: registry
      spec:
        replicas: 1
        selector:
          matchLabels:
            deployment: registry
        strategy:
          type: Recreate
        template:
          metadata:
            labels:
              deployment: registry
          spec:
            containers:
            - name: registry
              image: registry.hub.docker.com/library/registry:2.6.1
              imagePullPolicy: IfNotPresent
              ports:
              - containerPort: 5000
                protocol: TCP
              env:
              - name: REGISTRY_STORAGE_DELETE_ENABLED
                value: "true"
    - apiVersion: v1
```

```
    kind: Service
    metadata:
      name: registry
    spec:
      type: ClusterIP
      ports:
      - port: 80
        targetPort: 5000
      selector:
        deployment: registry
```

For namespaced resources, it is not necessary to enter the `namespace` field of the resource `metadata`. When the `namespace` field is not present, the resource is created within the session namespace for that workshop instance.

When resources are created, owner references are added, making the `WorkshopSession` custom resource corresponding to the workshop instance the owner. This means that when the workshop instance is deleted, any resources are deleted.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `session_id`: A unique ID for the workshop instance within the workshop environment.
- `session_namespace`: The namespace you create for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances. It is also the namespace where the service account that the workshop instance runs.
- `service_account`: The name of the service account the workshop instance runs as and which has access to the namespace you create for that workshop instance.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) you use for ingress routes and create for workshops.

The syntax for referencing the parameter is `$(parameter_name)`.

For cluster-scoped resources, you must set the name of the created resource so that it embeds the value of `$(session_namespace)`. This way the resource name is unique to the workshop instance, and you do not get a clash with a resource for a different workshop instance.

For examples of making use of the available parameters, see the following sections.

Overriding default role-based access control (RBAC) rules

By default the service account created for the workshop instance has `admin` role access to the session namespace created for that workshop instance. This enables the service account to be used to deploy applications to the session namespace and manage secrets and service accounts.

Where a workshop doesn't require `admin` access for the namespace, you can reduce the level of access it has to `edit` or `view` by setting the `session.namespaces.role` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
```

```
metadata:
  name: lab-role-testing
spec:
  title: Role Testing
  description: Play area for testing roles
  content:
    files: github.com/eduk8s-tests/lab-role-testing
  session:
    namespaces:
      role: view
```

To add additional roles to the service account, such as working with custom resource types added to the cluster, you can add the appropriate `Role` and `RoleBinding` definitions to the `session.objects` field described previously:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-kpack-testing
spec:
  title: Kpack Testing
  description: Play area for testing kpack
  content:
    files: github.com/eduk8s-tests/lab-kpack-testing
  session:
    objects:
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: Role
      metadata:
        name: kpack-user
      rules:
      - apiGroups:
        - kpack.io
        resources:
        - builds
        - builders
        - images
        - sourceresolvers
        verbs:
        - get
        - list
        - watch
        - create
        - delete
        - patch
        - update
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
      metadata:
        name: kpack-user
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: Role
        name: kpack-user
      subjects:
      - kind: ServiceAccount
        namespace: $(workshop_namespace)
        name: $(service_account)
```

Because the subject of a `RoleBinding` must specify the service account name and namespace it is contained within, both of which are unknown in advance, references to parameters for the workshop namespace and service account for the workshop instance are used when defining the subject.

You can add additional resources with `session.objects` to grant cluster-level roles and the service account `cluster-admin` role:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-admin-testing
spec:
  title: Admin Testing
  description: Play area for testing cluster admin
  content:
    files: github.com/eduk8s-tests/lab-admin-testing
  session:
    objects:
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRoleBinding
      metadata:
        name: $(session_namespace)-cluster-admin
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: cluster-admin
      subjects:
      - kind: ServiceAccount
        namespace: $(workshop_namespace)
        name: $(service_account)
```

In this case, the name of the cluster role binding resource embeds `$(session_namespace)` so that its name is unique to the workshop instance and doesn't overlap with a binding for a different workshop instance.

Running user containers as root

In addition to RBAC, which controls what resources a user can create and work with, Pod security policies are applied to restrict what Pods/containers a user deploys can do.

By default the deployments that a workshop user can create are allowed only to run containers as a non-root user. This means that many container images available on registries such as Docker Hub cannot be used.

If you are creating a workshop where a user must run containers as the root user, you must override the default `nonroot` security policy and select the `anyuid` security policy by using the `session.namespaces.security.policy` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-policy-testing
spec:
  title: Policy Testing
  description: Play area for testing security policies
  content:
    files: github.com/eduk8s-tests/lab-policy-testing
  session:
    namespaces:
      security:
        policy: anyuid
```

This setting applies to the primary session namespace and any secondary namespaces created.

Creating additional namespaces

For each workshop instance, a primary session namespace is created. You can deploy or pre-deploy applications into this namespace as part of the workshop.

If you need more than one namespace per workshop instance, you can create secondary namespaces in a couple of ways.

If the secondary namespaces are to be created empty, you can list the details of the namespaces under the property `session.namespaces.secondary`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    namespaces:
      role: admin
      budget: medium
      secondary:
      - name: $(session_namespace)-apps
        role: edit
        budget: large
        limits:
          default:
            memory: 512mi
```

When secondary namespaces are created, by default, the role, resource quotas, and limit ranges are set the same as the primary session namespace. Each namespace has a separate resource budget and it is not shared.

If required, you can override what `role`, `budget`, and `limits` are applied within the entry for the namespace.

Similarly, you can override the security policy for secondary namespaces on a case-by-case basis by adding the `security.policy` setting under the entry for the secondary namespace.

To create resources in the namespaces you create, create the namespaces by adding an appropriate `Namespace` resource to `session.objects` with the definitions of the resources you want to create in the namespaces:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    objects:
    - apiVersion: v1
      kind: Namespace
      metadata:
        name: $(session_namespace)-apps
```

When listing any other resources to be created within the added namespace, such as deployments, ensure that the `namespace` is set in the `metadata` of the resource. For example, `$(session_namespace)-apps`.

To override what role the service account for the workshop instance has in the added namespace, you can set the `learningcenter.tanzu.vmware.com/session.role` annotation on the `Namespace` resource:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    objects:
    - apiVersion: v1
      kind: Namespace
      metadata:
        name: $(session_namespace)-apps
        annotations:
          learningcenter.tanzu.vmware.com/session.role: view
```

To have a different resource budget set for the additional namespace, you can add the annotation `learningcenter.tanzu.vmware.com/session.budget` in the `Namespace` resource metadata and set the value to the required resource budget:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    objects:
    - apiVersion: v1
      kind: Namespace
      metadata:
        name: $(session_namespace)-apps
        annotations:
          learningcenter.tanzu.vmware.com/session.budget: large
```

To override the limit range values applied corresponding to the budget applied, you can add annotations starting with `learningcenter.tanzu.vmware.com/session.limits.` for each entry:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-namespace-testing
spec:
  title: Namespace Testing
  description: Play area for testing namespaces
  content:
    files: github.com/eduk8s-tests/lab-namespace-testing
  session:
    objects:
    - apiVersion: v1
      kind: Namespace
      metadata:
        name: $(session_namespace)-apps
        annotations:
```

```
          learningcenter.tanzu.vmware.com/session.limits.min.cpu: 50m
          learningcenter.tanzu.vmware.com/session.limits.min.memory: 32Mi
          learningcenter.tanzu.vmware.com/session.limits.max.cpu: 1
          learningcenter.tanzu.vmware.com/session.limits.max.memory: 1Gi
          learningcenter.tanzu.vmware.com/session.limits.defaultrequest.cpu: 50m
          learningcenter.tanzu.vmware.com/session.limits.defaultrequest.memory: 128Mi
          learningcenter.tanzu.vmware.com/session.limits.request.cpu: 500m
          learningcenter.tanzu.vmware.com/session.limits.request.memory: 1Gi
```

You only must supply annotations for the values you want to override.

If you need more fine-grained control over the limit ranges and resource quotas, set the value of the annotation for the budget to `custom` and add the `LimitRange` and `ResourceQuota` definitions to `session.objects`.

In this case you must set the `namespace` for the `LimitRange` and `ResourceQuota` resource to the name of the namespace, e.g., `$(session_namespace)-apps` so they are only applied to that namespace.

To set the security policy for a specific namespace other than the primary session namespace, you can add the annotation `learningcenter.tanzu.vmware.com/session.security.policy` in the `Namespace` resource metadata and set the value to `nonroot`, `anyuid`, or `custom` as necessary.

Shared workshop resources

Adding a list of resources to `session.objects` causes the given resources to be created for each workshop instance, whereas namespaced resources default to being created in the session namespace for a workshop instance.

If instead you want to have one common shared set of resources created once for the whole workshop environment, that is, used by all workshop instances, you can list them in the `environment.objects` field.

This might, for example, be used to deploy a single container image registry used by all workshop instances, with a Kubernetes job used to import a set of images into the container image registry, which are then referenced by the workshop instances.

For namespaced resources, it is not necessary to enter the `namespace` field of the resource `metadata`. When the `namespace` field is not present, the resource is created within the workshop namespace for that workshop environment.

When resources are created, owner references are added, making the `WorkshopEnvironment` custom resource correspond to the workshop environment of the owner. This means that when the workshop environment is deleted, any resources are also deleted.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `workshop_name`: The name of the workshop. This is the name of the `Workshop` definition the workshop environment was created against.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `environment_token`: The value of the token that must be used in workshop requests against the workshop environment.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances, and their service accounts, are created. It is the same namespace that shared workshop resources are created.

- `service_account`: The name of a service account you can use when creating deployments in the workshop namespace.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) used for ingress routes created for workshops.
- `ingress_secret`: The name of the ingress secret stored in the workshop namespace when secure ingress is used.

To create additional namespaces associated with the workshop environment, embed a reference to `$(workshop_namespace)` in the name of the additional namespaces with an appropriate suffix. Be careful that the suffix doesn't overlap with the range of session IDs for workshop instances.

When creating deployments in the workshop namespace, set the `serviceAccountName` of the `Deployment` resource to `$(service_account)`. This ensures the deployment makes use of a special Pod security policy set up by the Learning Center. If this isn't used and the cluster imposes a more strict default Pod security policy, your deployment might not work, especially if any image runs as `root`.

Workshop pod security policy

The pod for the workshop session is set up with a pod security policy that restricts what you can do from containers in the pod. The nature of the applied pod security policy is adjusted when enabling support for doing Docker builds. This in turn enables Docker builds inside the sidecar container attached to the workshop container.

If you are customizing the workshop by patching the pod specification using `session.patches` to add your own sidecar container, and that sidecar container must run as the root user or needs a custom pod security policy, you must override the default security policy for the workshop container.

To allow a sidecar container to run as the root user with no extra privileges required, you can override the default `nonroot` security policy and set it to `anyuid`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-policy-testing
spec:
  title: Policy Testing
  description: Play area for testing security policies
  content:
    files: github.com/eduk8s-tests/lab-policy-testing
  session:
    security:
      policy: anyuid
```

This is a different setting than described previously for changing the security policy for deployments made by a workshop user to the session namespaces. This setting applies only to the workshop container itself.

If you need more fine-grained control of the security policy, you must provide your own resources for defining the Pod security policy and map it so it is used. The details of the pod security policy must be in `environment.objects` and mapped by definitions added to `session.objects`. For this to be used, you must deactivate the application of the inbuilt pod security policies. You can do this by setting `session.security.policy` to `custom`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
```

```
metadata:
  name: lab-policy-testing
spec:
  title: Policy Testing
  description: Play area for testing policy override
  content:
    files: github.com/eduk8s-tests/lab-policy-testing
  session:
    security:
      policy: custom
    objects:
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
      metadata:
        namespace: $(workshop_namespace)
        name: $(session_namespace)-podman
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: $(workshop_namespace)-podman
      subjects:
      - kind: ServiceAccount
        namespace: $(workshop_namespace)
        name: $(service_account)
  environment:
    objects:
    - apiVersion: policy/v1beta1
      kind: PodSecurityPolicy
      metadata:
        name: aa-$(workshop_namespace)-podman
      spec:
        privileged: true
        allowPrivilegeEscalation: true
        requiredDropCapabilities:
        - KILL
        - MKNOD
        hostIPC: false
        hostNetwork: false
        hostPID: false
        hostPorts: []
        runAsUser:
          rule: MustRunAsNonRoot
        seLinux:
          rule: RunAsAny
        fsGroup:
          rule: RunAsAny
        supplementalGroups:
          rule: RunAsAny
        volumes:
        - configMap
        - downwardAPI
        - emptyDir
        - persistentVolumeClaim
        - projected
        - secret
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRole
      metadata:
        name: $(workshop_namespace)-podman
      rules:
      - apiGroups:
        - policy
        resources:
        - podsecuritypolicies
        verbs:
        - use
```

```
        resourceNames:
        - aa-$(workshop_namespace)-podman
```

By overriding the pod security policy, you are responsible for limiting what you can do from the workshop pod. In other words, add only the extra capabilities you need. The pod security policy is applied only to the pod the workshop session runs in. It does not change any pod security policy applied to service accounts that exist in the session namespace or other namespaces you have created.

There is a better way to set the priority of applied Pod security policies when a default Pod security policy is applied globally by mapping it to the `system:authenticated` group. This causes priority falling back to the order of the names of the Pod security policies. VMware recommends you use `aa-` as a prefix to the custom Pod security name you create. This ensures it takes precedence over any global default Pod security policy such as `restricted`, `pks-restricted` or `vmware-system-tmc-restricted`, no matter what the name of the global policy default.

Custom security policies for user containers

You can also set the value of the `session.namespaces.security.policy` setting as `custom`. This gives you more fine-grained control of the security policy applied to the pods and containers that a user deploys during a session. In this case you must provide your own resources that define and map the pod security policy.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-policy-testing
spec:
  title: Policy Testing
  description: Play area for testing policy override
  content:
    files: github.com/eduk8s-tests/lab-policy-testing
  session:
    namespaes:
      security:
        policy: custom
    objects:
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: RoleBinding
      metadata:
        namespace: $(workshop_namespace)
        name: $(session_namespace)-security-policy
      roleRef:
        apiGroup: rbac.authorization.k8s.io
        kind: ClusterRole
        name: $(workshop_namespace)-security-policy
      subjects:
      - kind: Group
        namespace: $(workshop_namespace)
        name: system:serviceaccounts:$(workshop_namespace)
  environment:
    objects:
    - apiVersion: policy/v1beta1
      kind: PodSecurityPolicy
      metadata:
        name: aa-$(workshop_namespace)-security-policy
      spec:
        privileged: true
        allowPrivilegeEscalation: true
        requiredDropCapabilities:
```

```
        - KILL
        - MKNOD
        hostIPC: false
        hostNetwork: false
        hostPID: false
        hostPorts: []
        runAsUser:
          rule: MustRunAsNonRoot
        seLinux:
          rule: RunAsAny
        fsGroup:
          rule: RunAsAny
        supplementalGroups:
          rule: RunAsAny
        volumes:
        - configMap
        - downwardAPI
        - emptyDir
        - persistentVolumeClaim
        - projected
        - secret
    - apiVersion: rbac.authorization.k8s.io/v1
      kind: ClusterRole
      metadata:
        name: $(workshop_namespace)-security-policy
      rules:
      - apiGroups:
        - policy
        resources:
        - podsecuritypolicies
        verbs:
        - use
        resourceNames:
        - aa-$(workshop_namespace)-security-policy
```

You can also do this on secondary namespaces by either changing the `session.namespaces.secondary.security.policy` setting to `custom` or using the `learningcenter.tanzu.vmware.com/session.security.policy: custom` annotation.

Defining additional ingress points

If running additional background applications, by default they are only accessible to other processes within the same container. For an application to be accessible to a user through their web browser, an ingress must be created mapping to the port for the application.

You can do this by supplying a list of the ingress points and the internal container port they map to by setting the `session.ingresses` field in the workshop definition:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      port: 8080
```

The form of the host name used in the URL to access the service is:

```
$(session_namespace)-application.$(ingress_domain)
```

This name cannot be `terminal`, `console`, `slides`, `editor`, or the name of any built-in dashboard. These values are reserved for the corresponding built-in capabilities providing those features.

In addition to specifying ingresses for proxying to internal ports within the same Pod, you can enter a `host`, `protocol` and `port` corresponding to a separate service running in the Kubernetes cluster:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      protocol: http
      host: service.namespace.svc.cluster.local
      port: 8080
```

You can use variables providing information about the current session within the `host` property if required:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      protocol: http
      host: service.$(session_namespace).svc.cluster.local
      port: 8080
```

Available variables are:

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances and where the service account that the workshop instance runs.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.

If the service uses standard `http` or `https` ports, you can leave out the `port` property, and the port is set based on the value of `protocol`.

When a request is proxied, you can specify additional request headers that must be passed to the service:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      protocol: http
      host: service.$(session_namespace).svc.cluster.local
      port: 8080
      headers:
      - name: Authorization
        value: "Bearer $(kubernetes_token)"
```

The value of a header can reference the following variable:

- `kubernetes_token`: The access token of the service account for the current workshop session, used for accessing the Kubernetes REST API.

Access controls enforced by the workshop environment or training portal protect accessing any service through the ingress. If you use the training portal, this must be transparent. Otherwise, supply any login credentials for the workshop again when prompted by your web browser.

External workshop instructions

In place of using workshop instructions provided with the workshop content, you can use externally hosted instructions instead. To do this set `sessions.applications.workshop.url` to the URL of an external web site:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      workshop:
        url: https://www.example.com/instructions
```

The external web site must displayed in an HTML iframe, is shown as is and must provide its own page navigation and table of contents if required.

The URL value can reference a number of predefined parameters. The available parameters are:

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.

- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where you create all deployments of the workshop instances and where the service account that the workshop instance runs.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) used for ingress routes that you create for workshops.

These could be used, for example, to reference workshops instructions hosted as part of the workshop environment:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      workshop:
        url: $(ingress_protocol)://$(workshop_namespace)-instructions.$(ingress_domai
n)
  environment:
    objects:
    - ...
```

In this case `environment.objects` of the workshop `spec` must include resources to deploy the application hosting the instructions and expose it through an appropriate ingress.

Deactivating workshop instructions

The aim of the workshop environment is to provide instructions for a workshop that users can follow. If you want instead to use the workshop environment as a development environment or as an administration console that provides access to a Kubernetes cluster, you can deactivate the display of workshop instructions provided with the workshop content. In this case, only the work area with the terminals, console, and so on, is displayed. To deactivate display of workshop instructions, add a `session.applications.workshop` section and set the `enabled` property to `false`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      workshop:
        enabled: false
```

Enabling the Kubernetes console

By default the Kubernetes console is not enabled. To enable it and make it available through the web browser when accessing a workshop, add a `session.applications.console` section to the workshop definition, and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      console:
        enabled: true
```

The Kubernetes dashboard provided by the Kubernetes project is used. To use Octant as the console, you can set the `vendor` property to `octant`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      console:
        enabled: true
        vendor: octant
```

When `vendor` is not set, `kubernetes` is assumed.

Enabling the integrated editor

By default the integrated web based editor is not enabled. To enable it and make it available through the web browser when accessing a workshop, add a `session.applications.editor` section to the workshop definition, and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      editor:
        enabled: true
```

The integrated editor used is based on Visual Studio Code. For more information about the editor, see https://github.com/cdr/code-server in GitHub.

To install additional VS Code extensions, do this from the editor. Alternatively, if building a custom workshop, you can install them from your `Dockerfile` into your workshop image by running:

```
code-server --install-extension vendor.extension
```

Replace `vendor.extension` with the name of the extension, where the name identifies the extension on the VS Code extensions marketplace used by the editor or provide a path name to a local `.vsix` file.

This installs the extensions into `$HOME/.config/code-server/extensions`.

If downloading extensions yourself and unpacking them or extensions are part of your Git repository, you can instead locate them in the `workshop/code-server/extensions` directory.

Enabling workshop downloads

You can provide a way for a workshop user to download files as part of the workshop content. Enable this by adding the `session.applications.files` section to the workshop definition and setting the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      files:
        enabled: true
```

The recommended way of providing access to files from workshop instructions is using the `files:download-file` clickable action block. This action ensures any file is downloaded to the local machine and is not displayed in the browser in place of the workshop instructions.

By default the user can access any files located under the home directory of the workshop user account. To restrict where the user can download files from, set the `directory` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      files:
        enabled: true
        directory: exercises
```

When the specified directory is a relative path, it is evaluated relative to the home directory of the workshop user.

Enabling the test examiner

The test examiner is a feature that allows a workshop to have verification checks that the workshop instructions can trigger. The test examiner is deactivated by default. To enable it, add a `session.applications.examiner` section to the workshop definition and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      examiner:
        enabled: true
```

You must provide any executable test programs for verification checks in the `workshop/examiner/tests` directory.

The test programs must return an exit status of 0 if the test is successful and nonzero if it fails. Test programs must not be persistent programs that can run forever.

Clickable actions for the test examiner are used within the workshop instructions to trigger the verification checks. You can configure them to start when the page of the workshop instructions is loaded.

Enabling session image registry

Workshops using tools such as `kpack` or `tekton` and which need a place to push container images when built can enable a container image registry. A separate registry is deployed for each workshop session.

The container image registry is currently fully usable only if workshops are deployed under a Learning Center Operator configuration that uses secure ingress. This is because a registry that is not secure is not trusted by the Kubernetes cluster as the source of container images when doing deployments.

To enable the deployment of a registry per workshop session, add a `session.applications.registry` section to the workshop definition and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      registry:
        enabled: true
```

The registry mounts a persistent volume for storing of images. By default the size of that persistent volume is 5Gi. To override the size of the persistent volume, add the `storage` property under the `registry` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      registry:
        enabled: true
        storage: 20Gi
```

The amount of memory provided to the registry defaults to 768Mi. To increase this, add the `memory` property under the `registry` section.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      registry:
        enabled: true
        memory: 1Gi
```

The registry is secured with a user name and password unique to the workshop session, and must be accessed over a secure connection.

To allow access from the workshop session, the file `$HOME/.docker/config.json` containing the registry credentials are injected into the workshop session. This is used by tools such as `docker`.

For deployments in Kubernetes, a secret of type `kubernetes.io/dockerconfigjson` is created in the namespace and applied to the `default` service account in the namespace. This means deployments made using the default service account can pull images from the registry without additional configuration. If creating deployments using other service accounts, add configuration to the service account or deployment to add the registry secret for pulling images.

If you need access to the raw registry host details and credentials, they are provided as environment variables in the workshop session. The environment variables are:

- `REGISTRY_HOST`: Contains the host name for the registry for the workshop session.
- `REGISTRY_AUTH_FILE`: Contains the location of the `docker` configuration file. Must be the equivalent of `$HOME/.docker/config.json`.
- `REGISTRY_USERNAME`: Contains the user name for accessing the registry.
- `REGISTRY_PASSWORD`: Contains the password for accessing the registry. This is different for each workshop session.
- `REGISTRY_SECRET`: Contains the name of a Kubernetes secret of type `kubernetes.io/dockerconfigjson` added to the session namespace, which contains the registry credentials.

The URL for accessing the registry adopts the HTTP protocol scheme inherited from the environment variable `INGRESS_PROTOCOL`. This is the same HTTP protocol scheme the workshop

sessions use.

To use any of the variables as data variables in workshop content, use the same variable name but in lowercase: `registry_host`, `registry_auth_file`, `registry_username`, `registry_password` and `registry_secret`.

Enabling ability to use Docker

To build container images in a workshop using `docker`, first enable it. Each workshop session is provided with its own separate Docker daemon instance running in a container.

Enabling support for running `docker` requires the use of a privileged container for running the Docker daemon. Because of the security implications of providing access to Docker with this configuration, VMware recommends that if you don't trust the people taking the workshop, any workshops that require Docker only be hosted in a disposable Kubernetes cluster that is destroyed at the completion of the workshop. You must not enable Docker for workshops hosted on a public service that is always kept running and where arbitrary users can access the workshops.

To enable support for using `docker` add a `session.applications.docker` section to the workshop definition and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      docker:
        enabled: true
```

The container that runs the Docker daemon mounts a persistent volume for storing of images which are pulled down or built locally. By default the size of that persistent volume is 5Gi. To override the size of the persistent volume, add the `storage` property under the `docker` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      docker:
        enabled: true
        storage: 20Gi
```

The amount of memory provided to the container running the Docker daemon defaults to 768Mi. To increase this, add the `memory` property under the `registry` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
```

```
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      docker:
        enabled: true
        memory: 1Gi
```

Access to the Docker daemon from the workshop session uses a local UNIX socket shared with the container running the Docker daemon. If it uses a local tool to access the socket connection for the Docker daemon directly rather than by running `docker`, it must use the `DOCKER_HOST` environment variable to set the location of the socket.

The Docker daemon is only available from within the workshop session and cannot be accessed outside of the pod by any tools deployed separately to Kubernetes.

Enabling WebDAV access to files

You can access or update local files within the workshop session from the terminal command line or editor of the workshop dashboard. The local files reside in the file system of the container the workshop session is running in.

To access the files remotely, you can enable WebDAV support for the workshop session.

To enable support for accessing files over WebDAV, add a `session.applications.webdav` section to the workshop definition, and set the `enabled` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      webdav:
        enabled: true
```

This causes a WebDAV server running within the workshop session environment. A set of credentials is also generated and are available as environment variables. The environment variables are:

- `WEBDAV_USERNAME`: Contains the user name that must be used when authenticating over WebDAV.
- `WEBDAV_PASSWORD`: Contains the password that must be used when authenticating over WebDAV.

To use any of the environment variables related to the container image registry as data variables in workshop content, declare this in the `workshop/modules.yaml` file in the `config.vars` section:

```
config:
  vars:
  - name: WEBDAV_USERNAME
  - name: WEBDAV_PASSWORD
```

The URL endpoint for accessing the WebDAV server is the same as the workshop session, with `/webdav/` path added. This can be constructed from the terminal using:

```
$INGRESS_PROTOCOL://$SESSION_NAMESPACE.$INGRESS_DOMAIN/webdav/
```

In workshop content it can be constructed using:

```
{{ingress_protocol}}://{{session_namespace}}.{{ingress_domain}}/webdav/
```

You can use WebDAV client support provided by your operating system or by using a standalone WebDAV client, such as CyberDuck.

Using WebDAV can make it easier to transfer files to or from the workshop session.

Customizing the terminal layout

By default a single terminal is provided in the web browser when accessing the workshop. If required, you can enable alternate layouts which provide additional terminals. To set the layout, add the `session.applications.terminal` section and include the `layout` property with the desired layout:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    applications:
      terminal:
        enabled: true
        layout: split
```

The options for the `layout` property are:

- `default`: Single terminal.
- `split`: Two terminals stacked above each other in ratio 60/40.
- `split/2`: Three terminals stacked above each other in ratio 50/25/25.
- `lower`: A single terminal is placed below any dashboard tabs, rather than being a tab of its own. The ratio of dashboard tab to terminal is 70/30.
- `none`: No terminal is displayed but can still be created from the drop down menu.

When adding the `terminal` section, you must include the `enabled` property and set it to `true` as it is a required field when including the section.

If you don't want a terminal displayed and also want to deactivate the ability to create terminals from the drop-down menu, set `enabled` to `false`.

Adding custom dashboard tabs

Exposed applications, external sites and additional terminals, can be given their own custom dashboard tab. This is done by specifying the list of dashboard panels and the target URL:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
```

```
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    ingresses:
    - name: application
      port: 8080
    dashboards:
    - name: Internal
      url: "$(ingress_protocol)://$(session_namespace)-application.$(ingress_domain)/"
    - name: External
      url: http://www.example.com
```

The URL values can reference a number of predefined parameters. The available parameters are:

- `session_namespace`: The namespace you create for and bind to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name`: The name of the workshop environment. Its current value is the name of the namespace for the workshop environment and subject to change.
- `workshop_namespace`: The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances you create and where the service account that the workshop instance runs.
- `ingress_domain`: The host domain under which you can create host names when creating ingress routes.
- `ingress_protocol`: The protocol (http/https) used for ingress routes that you create for workshops.

The URL can reference an external web site, however, that web site must not prohibit being embedded in an HTML iframe.

In the case of wanting to have a custom dashboard tab provide an additional terminal, the `url` property must use the form `terminal:<session>`, where `<session>` is replaced with the name of the terminal session. The name of the terminal session can be any name you choose, but must be restricted to lowercase letters, numbers, and dashes. You should avoid using numeric terminal session names such as “1”, “2”, and “3” as these are used for the default terminal sessions.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: Workshop
metadata:
  name: lab-application-testing
spec:
  title: Application Testing
  description: Play area for testing my application
  content:
    image: {YOUR-REGISTRY-URL}/lab-application-testing:main
  session:
    dashboards:
    - name: Example
      url: terminal:example
```

Configure the WorkshopEnvironment resource

This topic describes how you configure the `WorkshopEnvironment` custom resource, which defines a Learning Center workshop environment.

Specifying the workshop definition

Creating a workshop environment is performed as a separate step to loading the workshop definition. This allows multiple distinct workshop environments using the same workshop definition to be created if necessary.

To specify which workshop definition is to be used for a workshop environment, set the `workshop.name` field of the specification for the workshop environment.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
```

The workshop environment name specified in the workshop environment metadata does not need to be the same. It has to be different if you create multiple workshop environments from the same workshop definition.

When the workshop environment is created, the namespace created for the workshop environment uses the `name` specified in the `metadata`. This name is also used in the unique names of each workshop instance created under the workshop environment.

Overriding environment variables

A workshop definition can set a list of environment variables that must be set for all workshop instances. To override an environment variable specified in the workshop definition. or one defined in the container image, you can supply a list of environment variables as `session.env`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    env:
    - name: REPOSITORY-URL
      value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

You can use this to set the location of a back-end service, such as an image registry, used by the workshop.

Values of fields in the list of resource objects can reference several predefined parameters. The available parameters are:

- `session_` - A unique ID for the workshop instance within the workshop environment.
- `session_` - The namespace created for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_` - The name of the workshop environment. Currently, this is the same as the name of the namespace for the workshop environment. It is suggested that you do not rely on workshop environment name and namespace being the same, and use the most appropriate to cope with any future change.

- `workshop_` - The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances are created and where the workshop instance runs the service account exists.
- `service_` - The workshop instance service account's name and access to the namespace created for that workshop instance.
- `ingress_` - The host domain under which host names are created when creating ingress routes.
- `ingress_` - The protocol (http/https) used for ingress routes created for workshops.

The syntax for referencing one of the parameters is `$(parameter_name)`.

Overriding the ingress domain

To access a workshop instance using a public URL, you must specify an ingress domain. If an ingress domain is not specified, the default ingress domain that the Learning Center operator configured with is used.

When setting a custom domain, DNS must be configured with a wildcard domain to forward all requests for subdomains of the custom domain to the ingress router of the Kubernetes cluster.

To provide the ingress domain, you can set the `session.ingress.domain` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
```

By default, the workshop session is exposed using an HTTP connection if overriding the domain. If you require a secure HTTPS connection, you must have access to a wildcard SSL certificate for the domain. A secret of type `tls` must be created for the certificate in the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed. The name of that secret must then be set in the `session.ingress.secret` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
```

If HTTPS connections are terminated using an external load balancer and not by specifying a secret for ingresses managed by the Kubernetes ingress controller, then routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret by setting the `session.ingress.protocol` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
```

```
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      protocol: https
```

To override or set the ingress class, which dictates which ingress router is used when more than one option is available, you can add `session.ingress.class`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
      class: nginx
```

Controlling access to the workshop

By default, requesting a workshop using the `WorkshopRequest` custom resource is deactivated and must be enabled for a workshop environment by setting `request.enabled` to `true`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
```

With this enabled, anyone who can create a `WorkshopRequest` custom resource can request the creation of a workshop instance for the workshop environment.

To further control who can request a workshop instance in the workshop environment, you can first set an access token, which a user must know and supply with the workshop request. This is done by setting the `request.token` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
    token: lab-markdown-sample
```

The same name as the workshop environment is used in this example, which is probably not a good practice. Use a random value instead. The token value may be multiline.

As a second control measure, you can specify what namespaces the `WorkshopRequest` must be created. This means a user must have the specific ability to create `WorkshopRequest` resources in one of those namespaces.

You can specify the list of namespaces from which workshop requests for the workshop environment by setting `request.namespaces`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
    token: lab-markdown-sample
    namespaces:
    - default
```

To add the workshop namespace in the list, rather than list the literal name, you can reference a predefined parameter specifying the workshop namespace by including `$(workshop_namespace)`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    enabled: true
    token: lab-markdown-sample
    namespaces:
    - $(workshop_namespace)
```

Overriding the login credentials

When requesting a workshop using `WorkshopRequest`, a login dialog box is presented to the user when accessing the workshop instance URL. By default, the user name is `learningcenter`. The password is a random value the user must query from the `WorkshopRequest` status after creating the custom resource.

To override the user name, you can set the `session.username` field. To set the same fixed password for all workshop instances, you can set the `session.password` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  session:
    username: workshop
    password: lab-markdown-sample
```

Additional workshop resources

The workshop definition defined by the `Workshop` custom resource already declares a set of resources to be created with the workshop environment. You can use this when you have shared service applications the workshop needs, such as an container image registry or a Git repository server.

To deploy additional applications related to a specific workshop environment, you can declare them by adding them into the `environment.objects` field of the `WorkshopEnvironment` custom resource. You might use this deploy a web application used by attendees of a workshop to access their workshop instances.

For namespaced resources, it is not necessary to set the `namespace` field of the resource `metadata`. When the `namespace` field is not present, the resource is created within the workshop namespace for that workshop environment.

When resources are created, owner references are added, making the `WorkshopEnvironment` custom resource correspond to the owner of the workshop environment. This means that any resources are also deleted when the workshop environment is deleted.

Values of fields in the list of resource objects can reference several predefined parameters. The available parameters are:

- `workshop_` - The name of the workshop. This is the name of the `Workshop` definition the workshop environment was created against.
- `environment_` - The name of the workshop environment. Currently, this is the same as the name of the namespace for the workshop environment. Do not rely on the name and the workshop environment being the same, and use the most appropriate to cope with any future change.
- `environment_` - The token value must be used against the workshop environment in workshop requests.
- `workshop_` - The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances and their service accounts are created. It is the same namespace that shared workshop resources are created.
- `service_` - The service account name is used when creating deployments in the workshop namespace.
- `ingress_` - The host domain under which host names are created when creating ingress routes.
- `ingress_` - The protocol (http/https) used for ingress routes created for workshops.
- `ingress_` - The name of the ingress secret stored in the workshop namespace when secure ingress is being used.

To create additional namespaces associated with the workshop environment, embed a reference to `$(workshop_namespace)` in the name of the additional namespaces, with an appropriate suffix. Be mindful that the suffix doesn't overlap with the range of session IDs for workshop instances.

When creating deployments in the workshop namespace, set the `serviceAccountName` of the `Deployment` resource to `$(service_account)`. This ensures the deployment uses a special Pod security policy set up by the Learning Center. If this isn't used and the cluster imposes a more strict default Pod security policy, your deployment might not work, especially if any image expects to run as `root`.

Creation of workshop instances

After a workshop environment is created, you can create the workshop instances. You can request a workshop instance by using the `WorkshopRequest` custom resource. This can be a separate step,

or you can add them as resources under `environment.objects`.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopEnvironment
metadata:
  name: lab-markdown-sample
spec:
  workshop:
    name: lab-markdown-sample
  request:
    token: lab-markdown-sample
    namespaces:
    - $(workshop_namespace)
  session:
    username: learningcenter
    password: lab-markdown-sample
  environment:
    objects:
    - apiVersion: learningcenter.tanzu.vmware.com/v1beta1
      kind: WorkshopRequest
      metadata:
        name: user1
      spec:
        environment:
          name: $(environment_name)
          token: $(environment_token)
    - apiVersion: learningcenter.tanzu.vmware.com/v1beta1
      kind: WorkshopRequest
      metadata:
        name: user2
      spec:
        environment:
          name: $(environment_name)
          token: $(environment_token)
```

Using this method, the workshop environment is populated with workshop instances. You can query the workshop requests from the workshop namespace to discover the URLs for accessing each and the password if you didn't set one and a random password was assigned.

If you need more control over how the workshop instances were created using this method, you can use the `WorkshopSession` custom resource instead.

Configure the WorkshopRequest resource

This topic describes how you configure the `WorkshopRequest` custom resource, which defines a Learning Center workshop request.

Specifying workshop environment

The `WorkshopRequest` custom resource is used to request a workshop instance. It does not provide details needed to perform the deployment of the workshop instance. That information is sourced by the Learning Center Operator from the `WorkshopEnvironment` and `Workshop` custom resources.

The minimum required information in the workshop request is the name of the workshop environment. You supply this by setting the `environment.name` field.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample
```

```
spec:
  environment:
    name: lab-markdown-sample
```

A request is successful only if requesting a workshop instance for a workshop environment is enabled for that workshop. You can enable requests in the `WorkshopEnvironment` custom resource for the workshop environment.

If multiple workshop requests, for the same workshop environment or different ones, are created in the same namespace, the `name` defined in the `metadata` for the workshop request must be different for each. The value of this name is not used to name workshop instances. You need the `name` value to delete the workshop instance, which is done by deleting the workshop request.

Specifying required access token

If a workshop environment is configured to require an access token when making a workshop request against that environment, you can specify decide the token by setting the `environment.token` field.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopRequest
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
    token: lab-markdown-sample
```

Even with the token, the request fails if the following is true:

- The workshop environment has restricted the namespaces from which a workshop request was made
- The workshop request was not created in one of the permitted namespaces

Configure the TrainingPortal resource

This topic describes how you configure the `TrainingPortal` custom resource, which triggers the deployment of a set of Learning Center workshop environments and a set number of workshop instances.

Specifying the workshop definitions

You run multiple workshop instances to perform training to a group of people by creating the workshop environment and then creating each workshop instance. The `TrainingPortal` workshop resource bundles that up as one step.

Before creating the training environment, you must load the workshop definitions as a separate step.

To specify the names of the workshops to be used for the training, list them under the `workshops` field of the training portal specification. Each entry needs to define a `name` property, matching the name of the `Workshop` resource you created.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
```

```
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
  - name: lab-markdown-sample
```

When the training portal is created, it:

- Sets up the underlying workshop environments.
- Creates any workshop instances required to be created initially for each workshop.
- Deploys a web portal for attendees of the training to access their workshop instances.

Limit the number of sessions

When defining the training portal, you can set a limit on the workshop sessions that can be run concurrently. Set this limit by using the `portal.sessions.maximum` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
  - name: lab-markdown-sample
```

When you specify this, the maximum capacity of each workshop is set to the maximum value for the training portal as a whole. This means that any one workshop can have as many sessions running as specified by the maximum for the portal. However, to achieve this maximum for a given workshop, only instances of that workshop can be created. In other words, the maximum capacity can be spread across a number of workshops or it can be used in its entirety by a single workshop.

If you do not set `portal.sessions.maximum`, you must set the capacity for each individual workshop as detailed in the following section. In only setting the capacities of each workshop and not an overall maximum for sessions, you cannot share the overall capacity of the training portal across multiple workshops.

Capacity of individual workshops

When you have more than one workshop, you can want to limit how many instances of each workshop you can have so that they cannot grow to the maximum number of sessions for the whole training portal. This means you can stop a specific workshop from using all of the capacity of the training portal. To do this, set the `capacity` field under the entry for the workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
```

```
    capacity: 4
  - name: lab-markdown-sample
    capacity: 6
```

The value of `capacity` limits the number of workshop sessions for a specific workshop to that value. It must be less than or equal to the maximum number of workshops sessions for the portal, because the latter always sets the absolute limit.

Set reserved workshop instances

By default one instance of each of the listed workshops is created so when the initial user requests that workshop, it's available for use immediately.

When such a reserved instance is allocated to a user, provided that the workshop capacity hasn't been reached, a new instance of the workshop is created as a reserve ready for the next user. When a user ends a workshop and the workshop is at capacity, when the instance is deleted, a new reserve is created. The total of allocated and reserved sessions for a workshop cannot exceed the capacity for that workshop.

To override for a specific workshop how many reserved instances are kept on standby ready for users, you can set the `reserved` setting against the workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
    reserved: 2
  - name: lab-markdown-sample
    capacity: 6
    reserved: 4
```

You can set the value of `reserved` to 0 if you never want any reserved instances for a workshop and only want instances of that workshop created on demand when required for a user. Creating instances of a workshop on demand can result in a user waiting longer to access a workshop session.

When workshop instances are always created on demand, the oldest reserved instance is terminated to allow a new session of a desired workshop to be created. This also happens when reserved instances tie up capacity that could be used for a new session of another workshop. This occurs if any caps for specific workshops are met.

Override initial number of sessions

The initial number of workshop instances created for each workshop is specified by `reserved` or 1 if the setting hasn't been provided.

In the case where `reserved` is set in order to keep workshop instances on standby, you can indicate that initially you want more than the reserved number of instances created. This is useful when running a workshop for a set period of time. You might create up-front instances of the workshop corresponding to 75% of the expected number of attendees but with a smaller reserve number. With this configuration, new reserve instances only start to be created when the total number approaches 75% and all extra instances created up front have been allocated to users. This ensures

you have enough instances ready for when most people come, but you can also create other instances later if necessary:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: kubernetes-fundamentals
spec:
  portal:
    sessions:
      maximum: 100
  workshops:
  - name: lab-kubernetes-fundamentals
    initial: 75
    reserved: 5
```

Setting defaults for all workshops

If you have a list of workshops, and they all must be set with the same values for `capacity`, `reserved`, and `initial`, rather than add settings to each, you can set defaults to apply to all workshops under the `portal` section:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 10
    capacity: 6
    reserved: 2
    initial: 4
  workshops:
  - name: lab-asciidoc-sample
  - name: lab-markdown-sample
```

Set caps on individual users

By default a single user can run more than one workshop at a time. You can cap this to ensure that workshops run only one at a time. This prevents a user from wasting resources by starting more than one workshop and only working on one without shutting the other down.

To apply a limit on how many concurrent workshop sessions a user can start, use the `portal.sessions.registered` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
      registered: 1
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
    reserved: 2
  - name: lab-markdown-sample
```

```
    capacity: 6
    reserved: 4
```

This limit also applies to anonymous users when anonymous access is enabled through the training portal web interface or if sessions are being created through the REST API. To set a limit on anonymous users, you can set `portal.sessions.anonymous` instead:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: sample-workshops
spec:
  portal:
    sessions:
      maximum: 8
      anonymous: 1
  workshops:
  - name: lab-asciidoc-sample
    capacity: 4
    reserved: 2
  - name: lab-markdown-sample
    capacity: 6
    reserved: 4
```

Expiration of workshop sessions

After you reach the maximum capacity, no more workshops sessions can be created. After a workshop session is allocated to a user, it cannot be reassigned to another user.

If you are running a supervised workshop, set the capacity higher than the anticipated number of users in case you have more users than you expect. Use the setting for the reserved number of instances. This way, even if you set a higher capacity than needed, workshop sessions are only created as required and not all up front.

In supervised workshops, when the training is over, delete the whole training environment. All workshop sessions are then deleted.

To host a training portal over an extended period but don't know when users want to do a workshop, you can set up workshop sessions to expire after a set time. When expired, the workshop session is deleted and a new workshop session can be created in its place.

The maximum capacity is therefore the maximum at any one point in time, while the number can grow and shrink over time. So over an extended time, you can handle many more sessions than the set maximum capacity. The maximum capacity ensures you don't try to allocate more workshop sessions than you have resources for at a given time.

To set a maximum time allowed for a workshop session, use the `expires` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 8
    reserved: 1
    expires: 60m
```

The value needs to be an integer, followed by a suffix of 's', 'm' or 'h', corresponding to seconds, minutes, or hours.

The time period is calculated from when the workshop session is allocated to a user. When the time period is up, the workshop session is automatically deleted.

When an expiration period is specified, or when a user finishes a workshop or restarts the workshop, the workshop is also deleted.

To cope with users who claim a workshop session, but leave and don't use it, you can set a time period for when a workshop session with no activity is deemed orphaned and so is deleted. Do this using the `orphaned` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 8
    reserved: 1
    expires: 60m
    orphaned: 5m
```

Avoid this setting for supervised workshops where the whole event only lasts a certain length of time. This prevents a user's session from being deleted when the user takes breaks and the computer goes to sleep.

The `expires` and `orphaned` settings can also be set against `portal` to apply them to all workshops.

Updates to workshop environments

The list of workshops for an existing training portal can be changed by modifying the training portal definition applied to the Kubernetes cluster.

If you remove a workshop from the list of workshops, the workshop environment is marked as stopping and is deleted when all active workshop sessions have completed.

If you add a workshop to the list of workshops, a new workshop environment for it is created.

Changes to settings, such as the maximum number of sessions for the training portal or capacity settings for individual workshops, are applied to existing workshop environments.

By default a workshop environment is left unchanged if the corresponding workshop definition is changed. So in the default configuration, you must explicitly delete the workshop from the list of workshops managed by the training portal and then add it back again if the workshop definition changed.

If you prefer that workshop environments be replaced when the workshop definition changes, enable this by using the `portal.updates.workshop` setting:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    sessions:
      maximum: 8
    updates:
      workshop: true
  workshops:
  - name: lab-markdown-sample
    reserved: 1
```

```
    expires: 60m
    orphaned: 5m
```

When using this option, use the `portal.sessions.maximum` setting to limit the number of workshop sessions that can be run for the training portal as a whole. When replacing the workshop environment, the old workshop environment is retained if there is still an active workshop session being used. If the limit isn't set, the new workshop environment is still able to grow to its specific capacity and is not limited by how many workshop sessions are running against old instances of the workshop environment.

Overall, VMware recommends updating workshop environments when workshop definitions change only in development environments when working on workshop content. This is an especially good practice until you are familiar with how the training portal replaces existing workshop environments, and the resource implications of having old and new instances of a workshop environment running at the same time.

Override the ingress domain

To access a workshop instance using a public URL, specify an ingress domain. If an ingress domain isn't specified, the default ingress domain that the Learning Center Operator is configured with is used.

When setting a custom domain, DNS must have been configured with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.

To provide the ingress domain, set the `portal.ingress.domain` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

If overriding the domain, by default the workshop session is exposed using a HTTP connection. For a secure HTTPS connection, you must have access to a wildcard SSL certificate for the domain. A secret of type `tls` should be created for the certificate in the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed. The name of that secret must be set in the `portal.ingress.secret` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
      secret: learningcenter.tanzu.vmware.com-tls
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

You can terminate HTTPS connections by using an external load balancer instead of specifying a secret for ingresses managed by the Kubernetes ingress controller. In that case, when routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret. Instead, set the `portal.ingress.protocol` field:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
      protocol: https
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

To override or set the ingress class, which dictates which ingress router is used when more than one option is available, you can add `portal.ingress.class`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      domain: learningcenter.tanzu.vmware.com
      secret: learningcenter.tanzu.vmware.com-tls
      class: nginx
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

Override the portal host name

The default host name given to the training portal is the name of the resource with `-ui` suffix, followed by the domain specified by the resource or the default inherited from the configuration of the Learning Center Operator.

To override the generated host name, you can set `portal.ingress.hostname`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    ingress:
      hostname: labs
      domain: learningcenter.tanzu.vmware.com
      secret: learningcenter.tanzu.vmware.com-tls
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

This causes the host name to be `labs.learningcenter.tanzu.vmware.com` rather than the default generated name for this example of `lab-markdown-sample-ui.learningcenter.tanzu.vmware.com`.

Set extra environment variables

To override any environment variables for workshop instances created for a specific work, provide the environment variables in the `env` field of that workshop:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
    env:
    - name: REPOSITORY-URL
      value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

Values of fields in the list of resource objects can reference a number of predefined parameters. The available parameters are:

- `session_id` - A unique ID for the workshop instance within the workshop environment.
- `session_namespace` - The namespace created for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create its own resources.
- `environment_name` - The name of the workshop environment. For now this is the same as the name of the namespace for the workshop environment. Don't rely on them being the same, and use the most appropriate to cope with any future change.
- `workshop_namespace` - The namespace for the workshop environment. This is the namespace where all deployments of the workshop instances are created and where the service account that the workshop instance runs as exists.
- `service_account` - The name of the service account the workshop instance runs as and which has access to the namespace created for that workshop instance.
- `ingress_domain` - The host domain under which host names can be created when creating ingress routes.
- `ingress_protocol` - The protocol (http/https) used for ingress routes created for workshops.

The syntax for referencing one of the parameters is `$(parameter_name)`.

Override portal credentials

When a training portal is deployed, the user name for the admin and robot accounts uses the defaults of `learningcenter` and `robot@learningcenter`. The passwords for each account are randomly set.

For the robot account, the OAuth application client details used with the REST API are also randomly generated.

You can see what the credentials and client details are by running `kubectl describe` against the training portal resource. This will yield output that includes:

```
Status:
  learningcenter:
    Clients:
```

```
      Robot:
        Id:      ACZpcaLIT3qr725YWmXu8et9REl4HBg1
        Secret:  t5IfXbGZQThAKR43apoc9usOFVDv2BLE
    Credentials:
      Admin:
        Password:  0kGmMlYw46BZT2vCntyrRuFf1gQq5ohi
        Username:  learningcenter
      Robot:
        Password:  QrnY67ME9yGasNhq2OTbgWA4RzipUvo5
        Username:  robot@learningcenter
```

To override any of these values to set them to a predetermined value, you can add `credentials` and `clients` sections to the training portal specification.

To overload the credentials for the admin and robot accounts user:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    credentials:
      admin:
        username: admin-user
        password: top-secret
      robot:
        username: robot-user
        password: top-secret
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

To override the application client details for OAuth access by the robot account user:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    clients:
      robot:
        id: application-id
        secret: top-secret
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

Control registration type

By default the training portal web interface presents a registration page for users to create an account before selecting a workshop. If you want to allow only the administrator to log in, you can deactivate the registration page. Do this if you are using the REST API to create and allocate workshop sessions from a separate application:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
```

```
  portal:
    registration:
      type: one-step
      enabled: false
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

If rather than requiring users to register, you want to allow anonymous access, you can switch the registration type to anonymous:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
      type: anonymous
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

When a user visits the training portal home page in anonymous mode, an account for that user is automatically created and the user is logged in.

Specify an event access code

When deploying the training portal with anonymous access or open registration, anyone who knows the URL can access workshops. To at least restrict access to those who know a common event access code or password, you can set `portal.password`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    password: workshops-2020-07-01
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

When anonymous access is enabled and the training portal URL is accessed, users are asked to enter an event access code before they are redirected to the list of workshops or to the login page.

Make a list of workshops public

By default the index page providing the catalog of available workshop images is only available after a user has logged in, either through a registered account or as an anonymous user.

To make the catalog of available workshops public so they can be viewed before logging in, set the `portal.catalog.visibility` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
```

```
spec:
  portal:
    catalog:
      visibility: public
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

By default the catalog has visibility set to `private`. Use `public` to expose it.

This also makes it possible to access the list of available workshops from the catalog through the REST API, without authenticating against the REST API.

Use an external list of workshops

If you are using the training portal with registration deactivated, and you are using the REST API from a separate website to control creation of sessions, you can specify an alternate URL for providing the list of workshops.

This helps when the REST API creates a session and cookies are deleted or a session URL is shared with a different user. This means the value for the `index_url` supplied with the REST API request is lost.

To set the URL for the external site, use the `portal.index` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    index: https://www.example.com/
    registration:
      type: one-step
      enabled: false
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

If you supply this property, passing the `index_url` when creating a workshop session using the REST API is optional, and the value of this property is used. You can still supply `index_url` when using the REST API for a user to be redirected back to a sub-category of workshops on the site. The URL provided in the training portal definition then acts only as a fallback. That is, when the redirect URL becomes unavailable, it directs the user back to the top-level page for the external list of workshops.

If a user has logged into the training portal as the admin user, the user is not redirected to the external site and still sees the training portal's list of workshops.

Override portal title and logo

By default the web interface for the training portal displays a generic Learning Center logo and a page title of "Workshops." To override these, you can set `portal.title` and `portal.logo`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
```

```
  portal:
    title: Workshops
    logo: ....
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

The `logo` field should be a graphical image provided in embedded data URI format. The image is displayed with a fixed height of “40px”. The field can also be a URL for an image stored on a remote web server.

Allow the portal in an iframe

By default it is prohibited to display the web interface for the training portal in an iframe of another web site, because of content security policies applying to the training portal website.

To enable the ability to iframe the full training portal web interface or even a specific workshop session created using the REST API, provide the host name of the site that embeds it. Do this by using the `portal.theme.frame.ancestors` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    theme:
      frame:
        ancestors:
        - https://www.example.com
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

The property is a list of hosts, not a single value. To use a URL for the training portal in an iframe of a page, which, in turn, is embedded in another iframe of a page on a different site, list the host names of all sites.

Because the sites that embed iframes must be secure and use HTTPS, they cannot use plain HTTP. Browser policies prohibit promoting cookies to an insecure site when embedding using an iframe. If cookies cannot be stored, a user cannot authenticate against the workshop session.

Collect analytics on workshops

To collect analytics data on usage of workshops, supply a webhook URL. When you supply a webhook URL, events are posted to the webhook URL, including:

- Workshops started
- Pages of a workshop viewed
- Expiration of a workshop
- Completion of a workshop
- Termination of a workshop

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
```

```
metadata:
  name: lab-markdown-sample
spec:
  analytics:
    webhook:
      url: https://metrics.learningcenter.tanzu.vmware.com/?client=name&token=password
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

At present there is no metrics collection service compatible with the portal webhook reporting mechanism, so create a custom service or integrate it with any existing web front end for the portal REST API service.

If the collection service needs to be provided with a client ID or access token, it must accept using query string parameters set in the webhook URL.

Include the details of the event as HTTP POST data by using the `application/json` content type:

```
{
  "portal": {
    "name": "lab-markdown-sample",
    "uid": "91dfa283-fb60-403b-8e50-fb30943ae87d",
    "generation": 3,
    "url": "https://lab-markdown-sample-ui.learningcenter.tanzu.vmware.com"
  },
  "event": {
    "name": "Session/Started",
    "timestamp": "2021-03-18T02:50:40.861392+00:00",
    "user": "c66db34e-3158-442b-91b7-25391042f037",
    "session": "lab-markdown-sample-w01-s001",
    "environment": "lab-markdown-sample-w01",
    "workshop": "lab-markdown-sample",
    "data": {}
  }
}
```

When an event has associated data, it is included in the `data` dictionary:

```
{
  "portal": {
    "name": "lab-markdown-sample",
    "uid": "91dfa283-fb60-403b-8e50-fb30943ae87d",
    "generation": 3,
    "url": "https://lab-markdown-sample-ui.learningcenter.tanzu.vmware.com"
  },
  "event": {
    "name": "Workshop/View",
    "timestamp": "2021-03-18T02:50:44.590918+00:00",
    "user": "c66db34e-3158-442b-91b7-25391042f037",
    "session": "lab-markdown-sample-w01-s001",
    "environment": "lab-markdown-sample-w01",
    "workshop": "lab-markdown-sample",
    "data": {
      "current": "workshop-overview",
      "next": "setup-environment",
      "step": 1,
      "total": 4
    }
  }
}
```

The `user` field is the same portal user identity returned by the REST API when creating workshop sessions.

The event stream only produces events for things as they happen. For a snapshot of all current workshop sessions, use the REST API to request the catalog of available workshop environments, enabling the inclusion of current workshop sessions.

Track using Google Analytics

To record analytics data on usage of workshops by using Google Analytics, enable tracking by supplying a tracking ID for Google Analytics:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  analytics:
    google:
      trackingId: UA-XXXXXXX-1
  workshops:
  - name: lab-markdown-sample
    capacity: 3
    reserved: 1
```

Custom dimensions are used in Google Analytics to record details about the workshop a user is taking, including through which training portal and cluster it was accessed. So you can use the same Google Analytics tracking ID for multiple training portal instances running on different Kubernetes clusters.

To support use of custom dimensions in Google Analytics, configure the Google Analytics property with the following custom dimensions. They must be added in the order shown, because Google Analytics doesn't allow you to specify the index position for a custom dimension. It allocates them for you. You can't already have custom dimensions defined for the property, as the new custom dimensions must start at index of 1.

Custom Dimension Name	Index
workshop_name	1
session_namespace	2
workshop_namespace	3
training_portal	4
ingress_domain	5
ingress_protocol	6

In addition to custom dimensions against page accesses, events are also generated. These include:

- Workshop/Start
- Workshop/Finish
- Workshop/Expired

If you provide a Google Analytics tracking ID with the `TrainingPortal` resource definition, it takes precedence over the `SystemProfile` resource definition.

Note

Google Analytics is not a reliable way to collect data. Individuals or corporate firewalls can block the reporting of Google Analytics data. For more precise statistics, use the webhook URL for collecting analytics with a custom data collection platform.

Configure the SystemProfile resource

This topic describes how you use the `SystemProfile` custom resource to configure the Learning Center operator.

You can use the default system profile to set defaults for ingress and image pull secrets. You can also select an alternate profile for specific deployments if required.

Important

Changes made to the `SystemProfile` custom resource, or changes made by means of environment variables, don't take effect on already deployed `TrainingPortals`. You must recreate those for the changes to be applied. You only need to recreate the `TrainingPortal` resources, because this resource takes care of recreating the `WorkshopEnvironments` with the new values.

Operator default system profile

The Learning Center Operator, by default, uses an instance of the `SystemProfile` custom resource if it exists, named `default-system-profile`. You can override the name of the resource used by the Learning Center Operator as the default by setting the `SYSTEM_PROFILE` environment variable on the deployment for the Learning Center Operator. For example:

```
kubectl set env deployment/learningcenter-operator -e SYSTEM_PROFILE=default-system-pr
ofile -n learningcenter
```

The Learning Center Operator automatically detects and uses any changes to an instance of the `SystemProfile` custom resource. You do not need to redeploy the operator when changes are made.

Defining configuration for ingress

The `SystemProfile` custom resource replaces the use of environment variables to configure details such as the ingress domain, secret, and class.

Instead of setting `INGRESS_DOMAIN`, `INGRESS_SECRET`, and `INGRESS_CLASS` environment variables, create an instance of the `SystemProfile` custom resource named `default-system-profile`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  ingress:
    domain: learningcenter.tanzu.vmware.com
    secret: learningcenter.tanzu.vmware.com-tls
    class: nginx
```

If you terminate HTTPS connections by using an external load balancer and not by specifying a secret for ingresses managed by the Kubernetes ingress controller, then routing traffic into the

Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  ingress:
    domain: learningcenter.tanzu.vmware.com
    protocol: https
    class: nginx
```

Defining container image registry pull secrets

To work with custom workshop images stored in a private image registry, the system profile can define a list of image pull secrets. Add this to the service accounts used to deploy and run the workshop images. Set the `environment.secrets.pull` property to the list of secret names:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  environment:
    secrets:
      pull:
      - private-image-registry-pull
```

The secrets containing the image registry credentials must exist within the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed. The secret resources must be of type `kubernetes.io/dockerconfigjson`.

The secrets are added to the workshop namespace and are not visible to a user. No secrets are added to the namespace created for each workshop session.

Some container images are used as part of Learning Center itself, such as the container image for the training portal web interface and the builtin base workshop images. If you have copied these from the public image registries and stored them in a local private registry, use the `registry` section instead of the above setting. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  registry:
    secret: learningcenter-image-registry-pull
```

The `registry.secret` is the name of the secret containing the image registry credentials. This must be present in the `learningcenter` namespace or the namespace where the Learning Center Operator is deployed.

Defining storage class for volumes

Deployments of the training portal web interface and the workshop sessions make use of persistent volumes. By default the persistent volume claims do not specify a storage class for the volume. Instead, they rely on the Kubernetes cluster to specify a default storage class that works. If the

Kubernetes cluster doesn't define a suitable default storage class or you need to override it, you can set the `storage.class` property. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  storage:
    class: default
```

This only applies to persistent volume claims setup by the Learning Center Operator. If a user executes steps in a workshop that include making persistent volume claims, these are not automatically adjusted.

Defining storage group for volumes

The cluster must apply pod security policies where persistent volumes are used by Learning Center for the training portal web interface and workshop environments. These security policies ensure that permissions of persistent volumes are set correctly so they can be accessed by containers mounting the persistent volume. When the pod security policy admission controller is not enabled, the cluster institutes a fallback to enable access to volumes by enabling group access using the group ID of `0`.

In situations where the only class of persistent storage available is NFS or similar, you might have to override the group ID applied and set it to an alternate ID dictated by the file system storage provider. If this is required, you can set the `storage.group` property. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  storage:
    group: 1
```

Overriding the group ID to match the persistent storage relies on the group having write permission to the volume. If only the owner of the volume has permission, this does not work.

In this case, change the owner/group and permissions of the persistent volume such that the owner matches the user ID a container runs at. Alternatively, set the group to a known ID that is added as a supplemental group for the container and update the persistent volume to be writable to this group. This must be done by an `init` container running in the pod mounting the persistent volume.

To trigger this change of ownership and permissions, set the `storage.user` property. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  storage:
    user: 1
    group: 1
```

This results in:

- The `init` container running as the root user.
- The owner of the mount directory of the persistent volume being set to `storage.user`.

- The group being set to `storage.group`.
- The directory made group-writable.

The group is then added as the supplemental group to containers using the persistent volume. So they can write to the persistent volume, regardless of what user ID the container runs as. To that end, the specific value of `storage.user` doesn't matter, but you might need to set it to a specific user ID based on requirements of the storage provider.

Both these variations on the settings only apply to the persistent volumes used by Learning Center itself. If a workshop asks users to create persistent volumes, those instructions, or the resource definitions used, might need to be modified to work where the available storage class requires access as a specific user or group ID.

Further, the second method using the `init` container to fix permissions does not work if pod security policies are enforced. The ability to run a container as the root user is blocked in that case due to the restricted PSP, which is applied to workshop instances.

Restricting network access

Any processes running from the workshop container, and any applications deployed to the session namespaces associated with a workshop instance, can contact any network IP addresses accessible from the cluster. To restrict access to IP addresses or IP subnets, set `network.blockCIDRs`. This must be a CIDR block range corresponding to the subnet or a portion of a subnet you want to block. A Kubernetes `NetworkPolicy` is used to enforce the restriction. So the Kubernetes cluster must use a network layer supporting network policies, and the necessary Kubernetes controllers supporting network policies must be enabled when the cluster is installed.

If deploying to AWS, it is important to block access to the AWS endpoint for querying EC2 metadata, because it can expose sensitive information that workshop users should not haves access to. By default Learning Center will block the AWS endpoint on the TAP SystemProfile. If you need to replicate this block to other SystemProfiles, the configuration is as follows:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  network:
    blockCIDRs:
    - 169.254.169.254/32
    - fd00:ec2::254/128
```

Running Docker daemon rootless

If `docker` is enabled for workshops, Docker-in-Docker is run using a sidecar container. Because of the current state of running Docker-in-Docker and portability across Kubernetes environments, the `docker` daemon by default runs as `root`. Because a privileged container is also being used, this represents a security risk. Only run workshops requiring `docker` in disposable Kubernetes clusters or for users whom you trust.

You can partly mediate the risks of running `docker` in the Kubernetes cluster by running the `docker` daemon in rootless mode. However, not all Kubernetes clusters can support this due to the Linux kernel configuration or other incompatibilities.

To enable rootless mode, you can set the `dockerd.rootless` property to `true`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
```

```
metadata:
  name: default-system-profile
spec:
  dockerd:
    rootless: true
```

Use of `docker` can be made even more secure by avoiding the use of a privileged container for the `docker` daemon. This requires that you set up a specific configuration for nodes in the Kubernetes cluster. With this configuration, you can disallow the use of a privileged container by setting `dockerd.privileged` to `false`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    rootless: true
    privileged: false
```

For further details about the requirements for running rootless Docker-in-Docker and using a non-privileged container, see the Docker documentation.

Overriding network packet size

When you enable support for building container images using `docker` for workshops, because of network layering that occurs when doing `docker build` or `docker run`, you must adjust the network packet size (MTU) used for containers run from `dockerd` hosted inside the workshop container.

The default MTU size for networks is 1500, but, when containers are run in Kubernetes, the size available to containers is often reduced. To deal with this possibility, the MTU size used when `dockerd` is run for a workshop is set as 1400 instead of 1500.

You might need to override this value to an even lower value if you experience problems building or running images with `docker` support. These problems could include errors or timeouts in pulling images or when pulling software packages such as PyPi, npm, and so on.

To lower the value, set the `dockerd.mtu` property:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    mtu: 1400
```

To discover the maximum viable size, access the `docker` container run with a workshop and run `ifconfig eth0`. This yields something similar to:

```
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:07
          inet addr:172.17.0.7  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1350  Metric:1
          RX packets:270018 errors:0 dropped:0 overruns:0 frame:0
          TX packets:283882 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:86363656 (82.3 MiB)  TX bytes:65183730 (62.1 MiB)
```

If the `MTU` size is less than 1400, use the value given, or a smaller value, for the `dockerd.mtu` setting.

Image registry pull through cache

When running or building container images with `docker`, if the container image is hosted on Docker Hub, it is pulled down directly from the Docker Hub for each separate workshop session of that workshop.

Because the image is pulled from Docker Hub, this can be slow for all users, especially for large images. With Docker Hub introducing limits on how many images can be pulled anonymously from an IP address within a set period, this also can result in the cap on image pulls being reached. This prevents the workshop from being used until the period expires.

Docker Hub has a higher limit when pulling images as an authenticated user, but with the limit applied to the user rather than by IP address. For authenticated users with a paid plan on Docker Hub, there is no limit.

To attempt to avoid the impact of the limit, the first thing you can do is enable an image registry mirror with image pull-through. This is enabled globally and results in an instance of an image registry mirror being created in the workshop environment of workshops that enable `docker` support. This mirror is used for all workshops sessions created against that workshop environment. When the first user attempts to pull an image, it is pulled down from Docker Hub and cached in the mirror. Subsequent users are served up from the image registry mirror, avoiding the need to pull the image from Docker Hub again. Subsequent users also see a speed up in pulling the image, because the mirror is deployed to the same cluster.

To enable the use of an image registry mirror against Docker Hub, use:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    mirror:
      remote: https://registry-1.docker.io
```

For authenticated access to Docker Hub, create an access token under your Docker Hub account. Then set the `username` and `password` using the access token as the `password`. Do not use the password for the account itself. Using an access token makes it easier to revoke the token if necessary.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  dockerd:
    mirror:
      remote: https://registry-1.docker.io
      username: username
      password: access-token
```

An access token provides write access to Docker Hub. It is therefore also recommended you use a separate robot account in Docker Hub that is not used to host images and doesn't have write access to any other organizations. In other words, use it purely for reading images from Docker Hub.

If this is a free account, the higher limit on image pulls then applies. If the account is paid, there might be higher limits or no limit all all.

The image registry mirror is only used when running or building images using support for running `docker`. The mirror does not come into play when creating deployments in Kubernetes, which make use of images hosted on Docker Hub. Use of images from Docker Hub in deployments is still subject to the limit for anonymous access, unless you supply image registry credentials for the deployment so an authenticated user is used.

Setting default access credentials

When deploying a training portal using the `TrainingPortal` custom resource, the credentials for accessing the portal are unique for each instance. Find the details of the credentials by viewing status information added to the custom resources by using `kubectl describe`.

To override the credentials for the portals so the same set of credentials are used for each, add the desired values to the system profile.

To override the user name and password for the admin and robot accounts, use `portal.credentials`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  portal:
    credentials:
      admin:
        username: learningcenter
        password: admin-password
      robot:
        username: robot@learningcenter
        password: robot-password
```

To override the client ID and secret used for OAuth access by the robot account, use `portal.clients`:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  portal:
    clients:
      robot:
        id: robot-id
        secret: robot-secret
```

If the `TrainingPortal` has specified credentials or client information, they still take precedence over the values specified in the system profile.

Overriding the workshop images

When a workshop does not define a workshop image to use and instead downloads workshop content from GitHub or a web server, it uses the `base-environment` workshop image. The workshop content is then added to the container, overlaid on this image.

The version of the `base-environment` workshop image used is the most up-to-date, compatible version of the image available for that version of the Learning Center Operator when it was released.

If necessary you can override the version of the `base-environment` workshop image used by defining a mapping under `workshop.images`. For workshop images supplied as part of the Learning Center project, you can override the short names used to refer to them.

The short versions of the recognized names are:

- `base-environment:*` is a tagged version of the `base-environment` workshop image matched with the current version of the Learning Center Operator.

To override the version of the `base-environment` workshop image mapped to by the `*` tag, use:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  workshop:
    images:
      "base-environment:*": "registry.tanzu.vmware.com/learning-center/base-environmen
t:latest"
```

It is also possible to override where images are pulled from for any arbitrary image. This could be used where you want to cache the images for a workshop in a local image registry and avoid going outside of your network, or the cluster, to get them. This means you wouldn't need to override the workshop definitions for a specific workshop to change it. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  workshop:
    images:
      "{YOUR-REGISTRY-URL}/lab-k8s-fundamentals:main": "registry.test/lab-k8s-fundamen
tals:main"
```

Tracking using Google Analytics

If you want to record analytics data on usage of workshops using Google Analytics, you can enable tracking by supplying a tracking ID for Google Analytics. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  analytics:
    google:
      trackingId: UA-XXXXXXX-1
```

Custom dimensions are used in Google Analytics to record details about the workshop a user is taking and through which training portal and cluster it was accessed. You can therefore use the same Google Analytics tracking ID with Learning Center running on multiple clusters.

To support use of custom dimensions in Google Analytics, you must configure the Google Analytics property with the following custom dimensions. They must be added in the order shown, because Google Analytics doesn't allow you to specify the index position for a custom dimension and allocates them for you. You can't already have defined custom dimensions for the property, because the new custom dimensions must start at index of 1.

Custom Dimension Name	Index
workshop_name	1
session_namespace	2
workshop_namespace	3
training_portal	4
ingress_domain	5
ingress_protocol	6

In addition to custom dimensions against page accesses, events are also generated. These include:

- Workshop/Start
- Workshop/Finish
- Workshop/Expired

However, Google Analytics is not a reliable way to collect data. This is because individuals or corporate firewalls can block the reporting of Google Analytics data. For more precise statistics, use the webhook URL for collecting analytics with a custom data collection platform. Configuration of a webhook URL for analytics can only be specified on the `TrainingPortal` definition and cannot be specified globally on the `SystemProfile` configuration.

Overriding styling of the workshop

If using the REST API to create/manage workshop sessions, and the workshop dashboard is then embedded into an iframe of a separate site, you can perform minor styling changes of the dashboard, workshop content, and portal to match the separate site. To do this, provide CSS styles under `theme.dashboard.style`, `theme.workshop.style` and `theme.portal.style`. For dynamic styling or for adding hooks to report on progress through a workshop to a separate service, supply JavaScript as part of the theme under `theme.dashboard.script`, `theme.workshop.script`, and `theme.portal.script`. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: SystemProfile
metadata:
  name: default-system-profile
spec:
  theme:
    dashboard:
      script: |
        console.log("Dashboard theme overrides.");
      style: |
        body {
          font-family: "Comic Sans MS", cursive, sans-serif;
        }
    workshop:
      script: |
        console.log("Workshop theme overrides.");
      style: |
        body {
          font-family: "Comic Sans MS", cursive, sans-serif;
        }
    portal:
      script: |
        console.log("Portal theme overrides.");
      style: |
        body {
```

```
          font-family: "Comic Sans MS", cursive, sans-serif;
        }
```

Additional custom system profiles

If the default system profile is specified, it is used by all deployments managed by the Learning Center Operator, unless it was overridden by the system profile to use for a specific deployment. You can set the name of the system profile for deployments by setting the `system.profile` property of `TrainingPortal`, `WorkshopEnvironment`, and `WorkshopSession` custom resources. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  system:
    profile: learningcenter-tanzu-vmware-com-profile
  workshops:
  - name: lab-markdown-sample
    capacity: 1
```

Configure the WorkshopSession resource

This topic describes how you configure the `WorkshopSession` custom resource, which defines a Learning Center workshop session.

Specifying the session identity

When running training for multiple people, typically you'll use the `TrainingPortal` custom resource to set up a training environment. Alternatively, you can set up a workshop environment by using the `WorkshopEnvironment` custom resource, and then create requests for workshop instances by using the `WorkshopRequest` custom resource. If you're creating requests for workshop instances, and you need more control over how the workshop instances are set up, you can use `WorkshopSession` custom resource instead of `WorkshopRequest`.

To specify the workshop environment the workshop instance is created against, set the `environment.name` field of the specification for the workshop session. You must also specify the session ID for the workshop instance. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample-user1
spec:
  environment:
    name: lab-markdown-sample
  session:
    id: user1
```

The `name` of the workshop specified in the `metadata` of the training environment must be globally unique for the workshop instance you're creating. You must create a separate `WorkshopSession` custom resource for each workshop instance.

The session ID must be unique within the workshop environment that you're creating the workshop instance against.

Specifying the login credentials

You can control access to each workshop instance using login credentials. This ensures one workshop attendee cannot interfere with another.

To set login credentials for a workshop instance, set the `session.username` and `session.password` fields. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    username: learningcenter
    password: lab-markdown-sample
```

If you do not specify login credentials, the workshop instance has no access controls and anyone can access it.

Specifying the ingress domain

To access the workshop instance by using a public URL, you must specify an ingress domain. If an ingress domain isn't specified, use the default ingress domain that the Learning Center operator was configured with.

When setting a custom domain, configure DNS with a wildcard domain to forward all requests for sub-domains of the custom domain to the ingress router of the Kubernetes cluster.

To provide the ingress domain, you can set the `session.ingress.domain` field. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
```

You can create a full host name for the session by prefixing the ingress domain with a host name constructed from the name of the workshop environment and the session ID.

If overriding the domain, by default, the workshop session is exposed by using a HTTP connection. If you require a secure HTTPS connection, you must have access to a wildcard SSL certificate for the domain.

You must create a secret of type `tls` for the certificate in the `learningcenter` namespace or in the namespace where the Learning Center operator is deployed. You must then set the name of that secret in the `session.ingress.secret` field. For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
```

```
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
```

You can terminate HTTPS connections by using an external load balancer rather than by specifying a secret for ingresses managed by the Kubernetes ingress controller. When routing traffic into the Kubernetes cluster as HTTP connections, you can override the ingress protocol without specifying an ingress secret by setting the `session.ingress.protocol` field.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      protocol: https
```

To override or set the ingress class, add `session.ingress.class`. This dictates which ingress router is used when more than one option is available.

For example:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample-user1
  session:
    ingress:
      domain: training.learningcenter.tanzu.vmware.com
      secret: training.learningcenter.tanzu.vmware.com-tls
      class: nginx
```

Setting the environment variables

To set the environment variables for the workshop instance, provide the environment variables in the `session.env` field.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: WorkshopSession
metadata:
  name: lab-markdown-sample
spec:
  environment:
    name: lab-markdown-sample
  session:
    id: user1
    env:
    - name: REPOSITORY-URL
      value: YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE
```

Where `YOUR-GIT-URL-FOR-LAB-MARKDOWN-SAMPLE` is the Git repository URL for `lab-markdown-sample`. For example, `{YOUR-GIT-REPO-URL}/lab-markdown-sample`.

Values of fields in the list of resource objects can reference a number of predefined parameters. Available parameters are:

- `session_id` is a unique ID for the workshop instance within the workshop environment.
- `session_namespace` is the namespace created for and bound to the workshop instance. This is the namespace unique to the session and where a workshop can create their own resources.
- `environment_name` is the name of the workshop environment. For now this is the same as the name of the namespace for the workshop environment. Don't rely on them being the same, and use the most appropriate to cope with any future change.
- `workshop_namespace` is the namespace for the workshop environment. This is the namespace where all deployments of the workshop instances are created, and where the service account that the workshop instance runs as exists.
- `service_account` is the name of the service account the workshop instance runs as, and which has access to the namespace created for that workshop instance.
- `ingress_domain` is the host domain under which host names can be created when creating ingress routes.
- `ingress_protocol` is the protocol (http/https) used for ingress routes created for workshops.

The syntax for referencing one of the parameters is `$(parameter_name)`.

If the workshop environment had specified a set of extra environment variables to be set for workshop instances, it is up to you to incorporate those in the set of environment variables you list under `session.env`. That is, anything listed in `session.env` of the `WorkshopEnvironment` custom resource of the workshop environment is ignored.

Enable anonymous access to a Learning Center training portal

This topic describes how you enable anonymous access to a Learning Center training portal. The REST API with client authentication provides a means to have the portal create and manage workshop sessions on your behalf but allow a separate system handle user authentication.

If you do not need to authenticate users but still want to provide your own front end from which users select a workshop, such as when integrating workshops into an existing web property, you can enable anonymous mode and redirect users to a URL for workshop session creation.

Note

Anonymous mode is only recommended for temporary deployments and not for a permanent web site providing access to workshops.

Enabling anonymous access

Set the registration type to `anonymous` to enable full anonymous access to the training portal:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
```

```
    type: anonymous
workshops:
...
```

Note

Users can still visit the training portal directly and view the catalog of available workshops, so instead of linking to the main page of the training portal, link from your custom index page to the individual links for creating each workshop.

Triggering workshop creation

Direct users' browsers to a URL that is specific to a workshop to trigger creation and allocation of the workshop.

The URL format looks like this:

```
TRAINING-PORTAL-URL/workshops/environment/NAME/create/?index_url=INDEX
```

Where:

- `NAME` is the name of the workshop environment corresponding to the workshop that you creates.
- `INDEX` is the URL of your custom index page that contains the workshops.

The user is redirected back to this index page when:

- a user completes the workshop
- an error occurs

When a user is redirected back to the index page, a query string parameter is supplied to display a banner or other indication about why the user was returned.

The name of the query string parameter is `notification` and the possible values are:

- `session-deleted` - Used when the workshop session is completed or restarted.
- `workshop-invalid` - Used when the name of the workshop environment created is invalid.
- `session-unavailable` - Used when capacity is reached and a workshop session cannot be created.
- `session-invalid` - Used when an attempt is made to access a session that doesn't exist. This can occur when the workshop dashboard is refreshed after the workshop session is expired and deleted.

Enable anonymous access to a Learning Center training portal

This topic describes how you enable anonymous access to a Learning Center training portal. The REST API with client authentication provides a means to have the portal create and manage workshop sessions on your behalf but allow a separate system handle user authentication.

If you do not need to authenticate users but still want to provide your own front end from which users select a workshop, such as when integrating workshops into an existing web property, you can enable anonymous mode and redirect users to a URL for workshop session creation.

Note

Anonymous mode is only recommended for temporary deployments and not for a permanent web site providing access to workshops.

Enabling anonymous access

Set the registration type to `anonymous` to enable full anonymous access to the training portal:

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: lab-markdown-sample
spec:
  portal:
    registration:
      type: anonymous
  workshops:
  ...
```

Note

Users can still visit the training portal directly and view the catalog of available workshops, so instead of linking to the main page of the training portal, link from your custom index page to the individual links for creating each workshop.

Triggering workshop creation

Direct users' browsers to a URL that is specific to a workshop to trigger creation and allocation of the workshop.

The URL format looks like this:

```
TRAINING-PORTAL-URL/workshops/environment/NAME/create/?index_url=INDEX
```

Where:

- `NAME` is the name of the workshop environment corresponding to the workshop that you creates.
- `INDEX` is the URL of your custom index page that contains the workshops.

The user is redirected back to this index page when:

- a user completes the workshop
- an error occurs

When a user is redirected back to the index page, a query string parameter is supplied to display a banner or other indication about why the user was returned.

The name of the query string parameter is `notification` and the possible values are:

- `session-deleted` - Used when the workshop session is completed or restarted.
- `workshop-invalid` - Used when the name of the workshop environment created is invalid.
- `session-unavailable` - Used when capacity is reached and a workshop session cannot be created.
- `session-invalid` - Used when an attempt is made to access a session that doesn't exist. This can occur when the workshop dashboard is refreshed after the workshop session is

expired and deleted.

Use the Learning Center workshop catalog

A single training portal can host one or more workshops. This topic describes how you use the workshop catalog to list the available workshops and get information about them using the REST API.

Listing available workshops

The URL sub path for accessing the list of available workshop environments is `/workshops/catalog/environments/`. When making the request, you must supply the access token in the HTTP `Authorization` header with type set as `Bearer`:

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/catalog/environments/
```

The JSON response looks like this:

```
{
  "portal": {
    "name": "learningcenter-tutorials",
    "uid": "9b82a7b1-97db-4333-962c-97be6b5d7ee0",
    "generation": 451,
    "url": "<training_portal_url>",
    "sessions": {
      "maximum": 10,
      "registered": 0,
      "anonymous": 0,
      "allocated": 0
    }
  },
  "environments": [
    {
      "name": "learningcenter-tutorials-w01",
      "state": "RUNNING",
      "workshop": {
        "name": "lab-et-self-guided-tour",
        "id": "15e5f1a569496f335049bb00c370ee20",
        "title": "Workshop Building Tutorial",
        "description": "A guided tour of how to build a workshop for your team's learn
ing center.",
        "vendor": "",
        "authors": [],
        "difficulty": "",
        "duration": "",
        "tags": [],
        "logo": "",
        "url": "<workshop_repository_url>"
      },
      "duration": 1800,
      "capacity": 10,
      "reserved": 0,
      "allocated": 0,
      "available": 0
    }
  ]
}
```

For each workshop listed under `environments`, where a field listed under `workshop` has the same name as appears in the `Workshop` custom resource, it has the same meaning. The `id` field is an

additional field that can uniquely identify a workshop based on the name of the workshop image, the Git repository for the workshop, or the website hosting the workshop instructions. The value of the `id` field does not rely on the name of the `Workshop` resource and must be the same if the same workshop details are used but the name of the `Workshop` resource is different.

The `duration` field provides the time in seconds after which the workshop environment expires. The value is `null` if there is no expiration time for the workshop.

The `capacity` field is the maximum number of workshop sessions that you can create for the workshop.

The `reserved` field indicates how many instances of the workshop are reserved as hot spares. These are used to service requests for a workshop session. If no reserved instances are available and capacity has not been reached, a new workshop session is created on demand.

The `allocated` field indicates how many workshop sessions are active and currently allocated to a user.

The `available` field indicates how many workshop sessions are available for immediate allocation. This is never more than the number of reserved instances.

Under `portal.sessions`, the `allocated` field indicates the total number of allocated sessions across all workshops hosted by the portal.

Where `maximum`, `registered`, and `anonymous` are nonzero, they are the limit on the number of workshops run.

- The `maximum` is the total number of workshop sessions that can be run by the portal across all workshops. If `allocated` for the whole portal has reached `maximum`, no more workshop sessions are created.
- The value of `registered` when nonzero indicates a cap on the number of workshop sessions a single registered portal user can have running at the one time.
- The value of `anonymous` when nonzero indicates a cap on the number of workshop sessions an anonymous user can have running at the one time. Anonymous users are users created as a result of the REST API being used or if anonymous access is enabled when the user accesses the portal through the web interface.

By default, only workshop environments currently marked with a `state` of `RUNNING` are returned, that is, those workshop environments which are taking new workshop session requests. If you also want to see the workshop environments which are currently in the process of being shut down, you must provide the `state` query string parameter to the REST API call and indicate which states workshop environments to return for.

```
curl -v -H "Authorization: Bearer <access-token>" \
https://lab-markdown-sample-ui.test/workshops/catalog/environments/?state=RUNNING&stat
e=STOPPING
```

You can include the `state` query string parameter more than once to see workshop environments in both `RUNNING` and `STOPPING` states.

If anonymous access to the list of workshop environments is enabled and you are not authenticated when using the REST API endpoint, only workshop environments in a running state are returned.

Use session management for your Learning Center workshops

This topic describes how you use the REST API endpoints for session management, which allows you to request a workshop session to be allocated.

Deactivating portal user registration

When you use the REST API to trigger creation of workshop sessions, VMware recommends that you deactivate user registration through the training portal web interface. This means that only the admin user is able to directly access the web interface for the training portal.

```
apiVersion: learningcenter.tanzu.vmware.com/v1beta1
kind: TrainingPortal
metadata:
  name: learningcenter-tutorials
spec:
  portal:
    registration:
      type: one-step
      enabled: false
  workshops:
  ...
```

Requesting a workshop session

The form of the URL sub path for requesting the allocation of a workshop environment by using the REST API is `/workshops/environment/<name>/request/`. The name segment must be replaced with the name of the workshop environment. When making the request, the access token must be supplied in the HTTP `Authorization` header with type set as `Bearer`:

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/environment/<name>/request/?index_url=https://hub.tes
t/
```

You can supply a query string parameter, `index_url`. When you restart the workshop session from the workshop environment web interface, the session is deleted and the user is redirected to the supplied URL. This URL is that of your front end web application that has requested the workshop session, allowing users to select a different workshop.

The value of the `index_url` is not available if session cookies are cleared or a session URL is shared with another user. In this case, a user is redirected back to the training portal URL instead. You can override the global default for this case by specifying the index URL as part of the `TrainingPortal` configuration.

When successful, the JSON response from the request is of the form:

```
{
    "name": "educaes-tutorials-w01-s001",
    "user": "8d2d0c8b-6ff5-4244-b136-110fd8d8431a",
    "url": "/workshops/session/learningcenter-tutorials-w01-s001/activate/?token=6UIW4
D8Bhf0egVmsEKYlaOcTywrpQJGi&index_url=https%3A%2F%2Fhub.test%2F",
    "workshop": "learningcenter-tutorials",
    "environment": "learningcenter-tutorials-w01",
    "namespace": "learningcenter-tutorials-w01-s001"
}
```

This includes the name of the workshop session, an ID for identifying the user, and both a URL path with an activation token and an index URL included as query string parameters.

Redirect the user's browser to this URL path on the training portal host. Accessing the URL activates the workshop session and then redirects the user to the workshop dashboard.

If the workshop session is not activated, which confirms allocation of the session, the session is deleted after 60 seconds.

When a user is redirected back to the URL for the index page, a query string parameter is supplied to give the reason the user is being returned. You can use this to display a banner or other indication as to why the user was returned.

The name of the query string parameter is `notification` and the possible values are:

- `session-deleted` - Used when the workshop session is completed or restarted.
- `workshop-invalid` - Used when the name of the workshop environment supplied while attempting to create the workshop is invalid.
- `session-unavailable` - Used when capacity is reached, and a workshop session cannot be created.
- `session-invalid` - Used when an attempt is made to access a session that doesn't exist. This can occur when the workshop dashboard is refreshed sometime after the workshop session expired and was deleted.

In prior versions, the name of the session was returned through the "session" property, whereas the "name" property is now used. To support older code using the REST API, the "session" property is still returned, but it is deprecated.

Associating sessions with a user

When the workshop session is requested, a unique user account is created in the training portal each time. You can identify this account by using the `user` identifier, which is returned in the response.

The front end using the REST API to create workshop sessions can track user activity so that the training portal associates all workshop sessions created by the same user. Supply the `user` identifier with subsequent requests by the same user in the request parameter:

```
curl -v -H "Authorization: Bearer <access-token>" \
https://lab-markdown-sample-ui.test/workshops/environment/<name>/request/?index_url=ht
tps://hub.test/&user=<user>
```

If the supplied ID matches a user in the training portal, the training portal uses it internally and returns the same value for `user` in the response.

When the user does match, and if there is already a workshop session allocated to the user for the workshop being requested, the training portal returns a link to the existing workshop session, rather than requesting that the user create a new workshop session.

If the user is not a match, possibly because the training portal was completely redeployed since the last time it was accessed, the training portal returns a new user identifier.

The first time you make a request to create a workshop session for a user where `user` is not supplied, you can optionally supply request parameters for the following to set these as the user details in the training portal.

- `email` - The email address of the user.
- `first_name` - The first name of the user.
- `last_name` - The last name of the user.

These details will be accessible through the admin pages of the training portal.

When sessions are associated with a user, you can query all active sessions for that user across the different workshops hosted by the instance of the training portal:

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/user/<user>/sessions/
```

The response is of the form:

```
{
  "user": "8d2d0c8b-6ff5-4244-b136-110fd8d8431a",
  "sessions": [
    {
      "name": "learningcenter-tutorials-w01-s001",
      "workshop": "learningcenter-tutorials",
      "environment": "learningcenter-tutorials-w01",
      "namespace": "learningcenter-tutorials-w01-s001",
      "started": "2020-07-31T03:57:33.942Z",
      "expires": "2020-07-31T04:57:33.942Z",
      "countdown": 3353,
      "extendable": false
    }
  ]
}
```

After a workshop has expired or has otherwise been shut down, the training portal no longer returns an entry for the workshop.

Listing all workshop sessions

To get a list of all running workshops sessions allocated to users, provide the `sessions=true` flag to the query string parameters of the REST API call. This lists the workshop environments available through the training portal.

```
curl -v -H "Authorization: Bearer <access-token>" |
<training-portal-url>/workshops/catalog/environments/?sessions=true
```

The JSON response is of the form:

```
{
  "portal": {
    "name": "learningcenter-tutorials",
    "uid": "9b82a7b1-97db-4333-962c-97be6b5d7ee0",
    "generation": 476,
    "url": "<training-portal-url>",
    "sessions": {
      "maximum": 10,
      "registered": 0,
      "anonymous": 0,
      "allocated": 1
    }
  },
  "environments": [
    {
      "name": "learningcenter-tutorials-w01",
      "state": "RUNNING",
      "workshop": {
        "name": "lab-et-self-guided-tour",
        "id": "15e5f1a569496f335049bb00c370ee20",
        "title": "Workshop Building Tutorial",
        "description": "A guided tour of how to build a workshop for your team's learn
ing center.",
        "vendor": "",
        "authors": [],
        "difficulty": "",
        "duration": "",
        "tags": [],
        "logo": "",
        "url": "<workshop-repository-url>"
      },
```

```
      "duration": 1800,
      "capacity": 10,
      "reserved": 0,
      "allocated": 1,
      "available": 0,
      "sessions": [
        {
          "name": "learningcenter-tutorials-w01-s002",
          "state": "RUNNING",
          "namespace": "learningcenter-tutorials-w01-s002",
          "user": "672338f3-4085-4782-8d9b-ae1637e1c28c",
          "started": "2021-11-05T15:50:04.824Z",
          "expires": "2021-11-05T16:20:04.824Z",
          "countdown": 1737,
          "extendable": false
        }
      ]
    }
  ]
}
```


No workshop sessions are returned if anonymous access to this REST API endpoint is enabled and you are not authenticated against the training portal.

Only workshop environments with a `state` of `RUNNING` are returned by default. To see workshop environments that are shut down and any workshop sessions that still haven't been completed, supply the `state` query string parameter with value `STOPPING`.

```
curl -v -H "Authorization: Bearer <access-token>" \
<training-portal-url>/workshops/catalog/environments/?sessions=true&state=RUNNING&stat
e=STOPPING
```

Include the `state` query string parameter more than once to see workshop environments in both `RUNNING` and `STOPPING` states.

Use client authentication for Learning Center

This topic describes how you can use the portal REST API to integrate access to workshops into an existing website or to create a custom web interface for accessing workshops hosted across one or more training portals.

The training portal web interface is a quick way of providing access to a set of workshops when running a supervised training workshop. The REST API gives you access to the list of workshops hosted by a training portal instance and allow you to request and access workshop sessions. This bypasses the training portal's own user registration and log in so you can implement whatever access controls you need. This can include anonymous access or access integrated into an organization's single sign-on system.

Querying the credentials

To provide access to the REST API, a robot account is automatically provisioned. Obtain the login credentials and details of the OAuth client endpoint used for authentication by querying the resource definition for the training portal after it is created and the deployment completed. If using `kubectl describe`, use:

```
kubectl describe trainingportal.learningcenter.tanzu.vmware.com/<training-portal-name>
```

The status section of the output reads:

```
Status:
  learningcenter:
    Clients:
      Robot:
        Id:      ACZpcaLIT3qr725YWmXu8et9REl4HBg1
        Secret:  t5IfXbGZQThAKR43apoc9usOFVDv2BLE
    Credentials:
      Admin:
        Password:  0kGmMlYw46BZT2vCntyrRuFf1gQq5ohi
        Username:  learningcenter
      Robot:
        Password:  QrnY67ME9yGasNhq2OTbgWA4RzipUvo5
        Username:  robot@learningcenter
```

Use the admin login credentials when you log in to the training portal web interface to access admin pages.

Use the robot login credentials if you want to access the REST API.

Requesting an access token

Before you can make requests against the REST API to query details about workshops or request a workshop session, you must log in through the REST API to get an access token.

This is done from any front-end web application or provisioning system, but the step is equivalent to making a REST API call by using `curl` of:

```
curl -v -X POST -H \
"Content-Type: application/x-www-form-urlencoded" \
-d "grant_type=password&username=robot@learningcenter&password=<robot-password>" \
-u "<robot-client-id>:<robot-client-secret>" \
<training-portal-url>/oauth2/token/
```

The URL sub path is `/oauth2/token/`.

Upon success, the output is a JSON response consisting of:

```
{
    "access_token": "tg31ied56fOo4axuhuZLHj5JpUYCEL",
    "expires_in": 36000,
    "token_type": "Bearer",
    "scope": "user:info",
    "refresh_token": "1ryXhXbNA9RsTRuCE8fDAyZToJmp30"
}
```

Refreshing the access token

The access token that is provided expires: it needs to be refreshed before it expires if in use by a long-running application.

To refresh the access token, use the equivalent of:

```
curl -v -X POST -H \
"Content-Type: application/x-www-form-urlencoded" \
-d "grant_type=refresh_token&refresh_token=<refresh-token>& \client_id=<robot-client-i
d>&client_secret=<robot-client-secret>" \
https://lab-markdown-sample-ui.test/oauth2/token/
```

As with requesting the initial access token, the URL sub path is `/oauth2/token/`.

The JSON response is of the same format as if a new token was requested.

Troubleshoot Learning Center

This topic gives you troubleshooting and recovery steps for Learning Center known issues.

Training portal stays in pending state

The training portal stays in a “pending” state.

The Training Portal custom resource (CR) has a status property. To see the status, run:

```
kubectl get trainingportals.learningcenter.tanzu.vmware.com
```

Explanation

If the status stays in a pending state, the TLS secret `tls` might not be available. Other errors can also cause the status to stay in a pending state, so it is important to check the operator and portal logs to execute the right steps.

Solution

1. Access the operator logs by running:

   ```
   kubectl logs deployment/learningcenter-operator -n learningcenter
   ```

   Access the portal logs by running:

   ```
   kubectl logs deployment/learningcenter-portal -n {PORTAL_NAMESPACE}
   ```

2. Check whether the TLS secret `tls` is available. The TLS secret must be on the Learning Center operator namespace (by default `learningcenter`). If the TLS secret is not on the Learning Center operator namespace, the operator logs contain the following error:

   ```
   ERROR:kopf.objects:Handler 'learningcenter' failed temporarily: TLS secret tls
   is not available
   ```

3. Follow the steps in Enforcing Secure Connections in *Learning Center Operator* to create the TLS secret.

4. Redeploy the `trainingPortal` resource.

image-policy-webhook-service not found

You are installing a Tanzu Application Platform profile and you get this error:

```
Internal error occurred: failed calling webhook "image-policy-webhook.signing.run.tanz
u.vmware.com": failed to call webhook: Post "https://image-policy-webhook-service.imag
e-policy-system.svc:443/signing-policy-check?timeout=10s": service "image-policy-webho
ok-service" not found
```

Explanation

This is a race condition error among some Tanzu Application Platform packages.

Solution

To recover from this error you only need to redeploy the trainingPortal resource.

Updates to Tanzu Application Platform values file not reflected in Learning Center Training Portal

If you installed Learning Center through Tanzu profiles, then your installation made use of a tap-values.yaml file where configurations were specified for Learning Center. If you make updates to these configurations using this command:

```
tanzu package installed update tap --package-name tap.tanzu.vmware.com --version {VERS
ION} -f tap-values.yml -n tap-install
```

then the changes are not reflected in the deployed Learning Center Training Portal resource. Tap package updates currently `DO NOT` update running Learning Center Training Portal resources.

Run one of these commands to validate changes made to parameters provided to the Learning Center Operator. These parameters include ingressDomain, TLS secret, ingressClass, and others.

Command:

```
kubectl describe systemprofile
```

Command:

```
kubectl describe pod  -n learningcenter
```

Explanation

By design, the training portal resources do not react to any changes on the parameters provided when the training portals were created. This prevents any change on the `trainingportal` resource from affecting any online user running a workshop.

Solution

You must restart the operator resource by first deleting the operator pod:

```
kubectl delete pod -n learningcenter learningcenter-operator-$OPERATOR_POD_NAME
```

Then delete the training portal resource. Redeploy `trainingportal` in a maintenance window where Learning Center is unavailable while the`systemprofile` is updated.

Increase your cluster's resources

If you don't have enough nodes or enough resources on nodes for deploying the workloads, node pressure might occur. In this case, follow your cloud provider's instructions on how to scale out or scale up your cluster.

Kubernetes Api Timeout error

The following operator error log means there is a connection error with the Kubernetes API server:

```
operator-log: unexpected error occurred. Read timed out.
```

This error has been found when running Learning Center with the Azure AkS cloud provider.

Solution

To fix this error:

1. Delete the operator pod on the learningcenter namespace.
2. Delete the training portal once the operator is running again by using:

```
kubectl delete trainingportals $PORTAL_NAME
```

1. Redeploy the `trainingPortal` resource.

No URL returned to your trainingportal

After deploying the Learning Center Operator and Trainingportal resources, the following command can yield the resource with no URL, even though your resources deployed correctly and are running:

```
kubectl get trainingportals
```

You also already specified learningcenter.mydomain.com in your tap values YAML file if installed through Tanzu Application Platform. See specifying ingress domain

Solution

Learning center requires that you use a wildcard domain (Wildcard DNS entry) to access your training portal in the browser. This configuration must be done in your DNS provider with a rule that points your wildcard domain to your IP/Load balancer.

For example, if using the default workshop on an Elastic Kubernetes Service (EKS) cluster, your URL could look something like:

`learning-center-guided.learningcenter.yourdomain.com`

Where learningcenter.yourdomain.com needs a DNS configuration made to point to your default ingress controller.

In this case, the wildcard domain configuration needed is `*.learningcenter.yourdomain.com`.

After this configuration is made, you might need to restart your operator resource by deleting and redeploying to see the URL update.

Overview of Namespace Provisioner

Namespace Provisioner provides a secure, automated way for you to provision namespaces with the resources and namespace-level privileges required for your workloads to function as intended.

Description

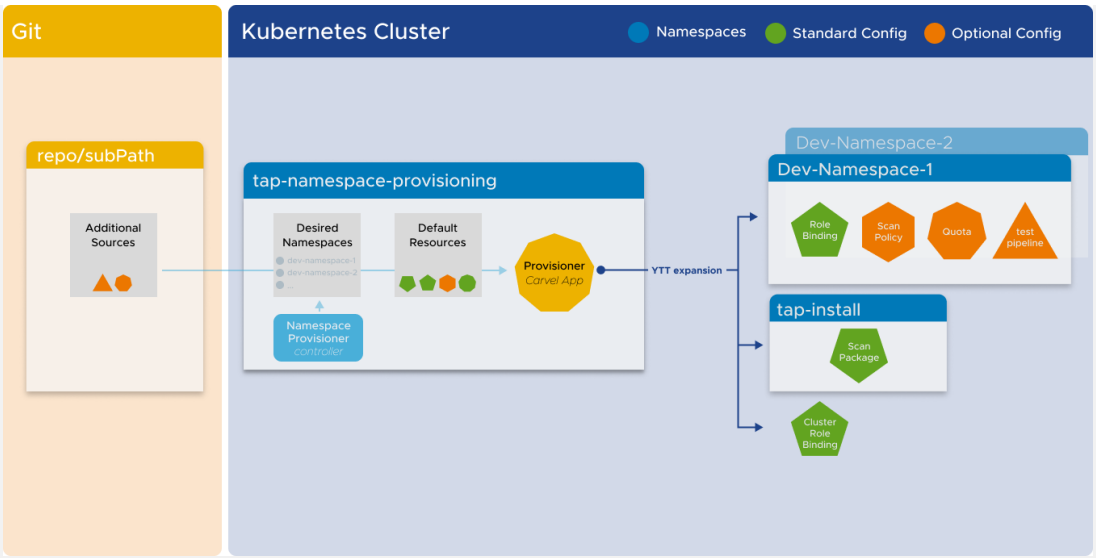
Namespace Provisioner enables platform operators to add additional customized namespace-scoped resources using GitOps to meet their organization's requirements and provides continuous reconciliation using the kapp-controller to maintain the desired state of the namespace-scoped resources.

Namespace Provisioner enables operators that are new to Kubernetes to automate the provisioning of multiple developer namespaces in a shared cluster. For organizations that have already adopted Kubernetes, Namespace Provisioner is also compatible with existing Kubernetes tooling.

Modes

Use Namespace Provisioner with one of the following modes:

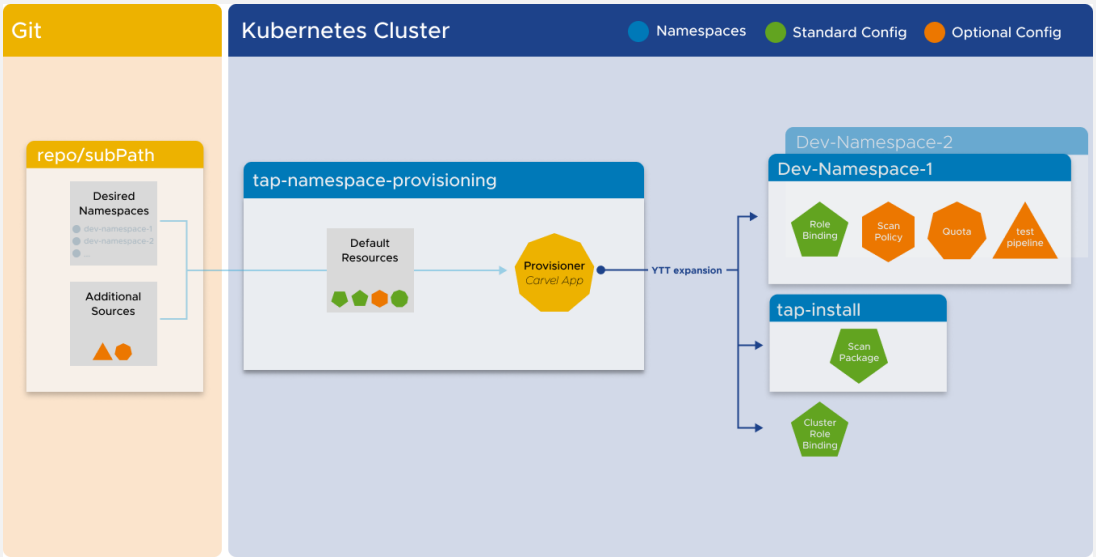
Controller mode
Controller mode has the following characteristics:



- The list of developer namespaces is managed by the Namespace Provisioner controller using a label selector `apps.tanzu.vmware.com/tap-ns=""`
- Namespace Provisioner creates default resources that are shipped Out of the Box in all managed namespaces.
- Namespace Provisioner creates additional Platform Operator templated resources stored in Git repository locations specified under the `additional_sources` section in Namespace Provisioner configuration. For more information, see Customize Installation of Namespace Provisioner.

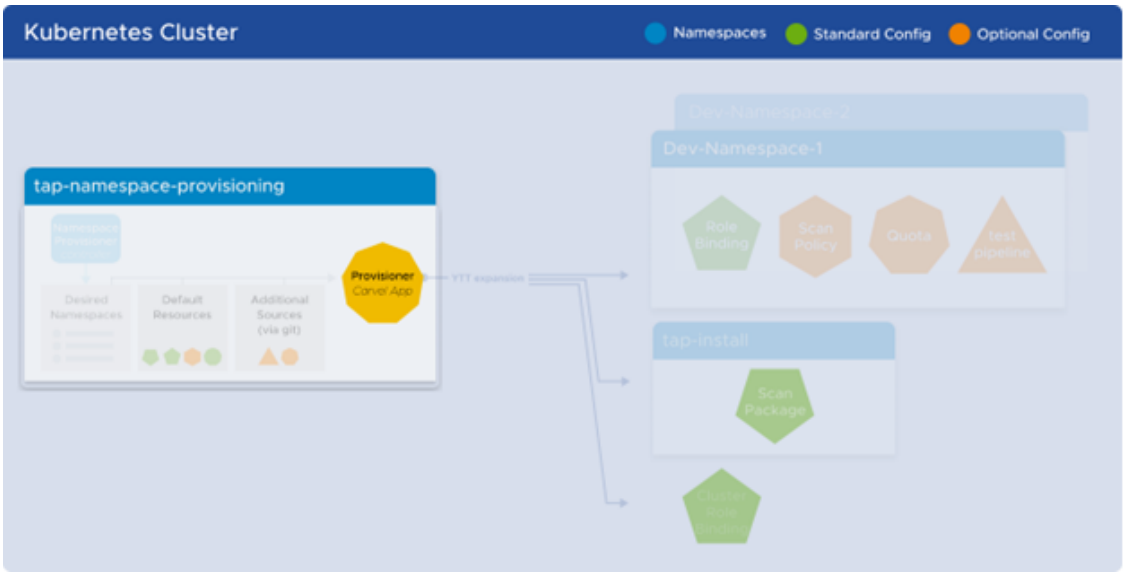
GitOps mode

Gitops mode has the following characteristics



- The list of developer namespaces is managed in a Git repository that is specified in the `gitops_install` section of the Namespace Provisioner configuration.
- Namespace Provisioner creates default resources that are shipped Out of the Box in all managed namespaces.
- Namespace Provisioner creates additional Platform Operator templated resources stored in Git repository locations specified under `additional_sources` in Namespace Provisioner configuration. For more information, see Customize Installation of Namespace Provisioner.

Provisioner Carvel application



Namespace Provisioner consists of a Carvel application called `provisioner` that facilitates the creation of resources in the managed developer namespaces. The `provisioner` application uses ytt to templatize a set of resources into installations in multiple namespaces.

Desired namespaces

The following section describes how the list of desired developer namespaces is managed in controller and GitOps modes.

Controller mode

In controller mode, the list of desired namespaces used by the `provisioner` application to create resources in, is maintained in the `desired-namespaces` ConfigMap. This ConfigMap is managed by the Namespace Provisioner controller and it provides a declarative way to indicate which namespaces should be populated with resources. The ConfigMap consists of a list of namespace objects, with a required `name` parameter, and optional additional parameters which are used as `data.values` for customizing defined resources.

For example,

```
---
apiVersion: v1
kind: ConfigMap
metadata:
    name: desired-namespaces
    namespace: tap-namespace-provisioning
    annotations:
        kapp.k14s.io/create-strategy: fallback-on-update
        namespace-provisioner.apps.tanzu.vmware.com/no-overwrite: "" #! This annotat
ion tells the provisioner app to not override this configMap as this is your desired
state.
data:
    namespaces.yaml: |
        #@data/values
        ---
        namespaces:
        - name: dev-ns1
        # additional parameters about dev-ns1 added via label/annotations or GitOps
        - name: dev-ns2
        # additional parameters about dev-ns1 added via label/annotations or GitOps
```

GitOps mode
In the GitOps mode, the list of desired namespaces used by the `provisioner` application to create resources in, is maintained in a Git repository as a ytt data values file as shown in this sample file. This file provides a declarative way to indicate which namespaces should be populated with resources. For more information, see the `Options if using GitOps` section in Customize Install.

Namespace Provisioner controller

The Namespace Provisioner controller (controller) is installed by default and manages the content contained in the `desired-namespaces` ConfigMap. The controller watches namespaces in the cluster and updates the `desired-namespaces` ConfigMap with a list of all namespaces that match the namespace label selector.The defalut namespace label selector is `apps.tanzu.vmware.com/tap-ns`. For more information, see Use a different label selector than default.

Overview of Namespace Provisioner

Namespace Provisioner provides a secure, automated way for you to provision namespaces with the resources and namespace-level privileges required for your workloads to function as intended.

Description

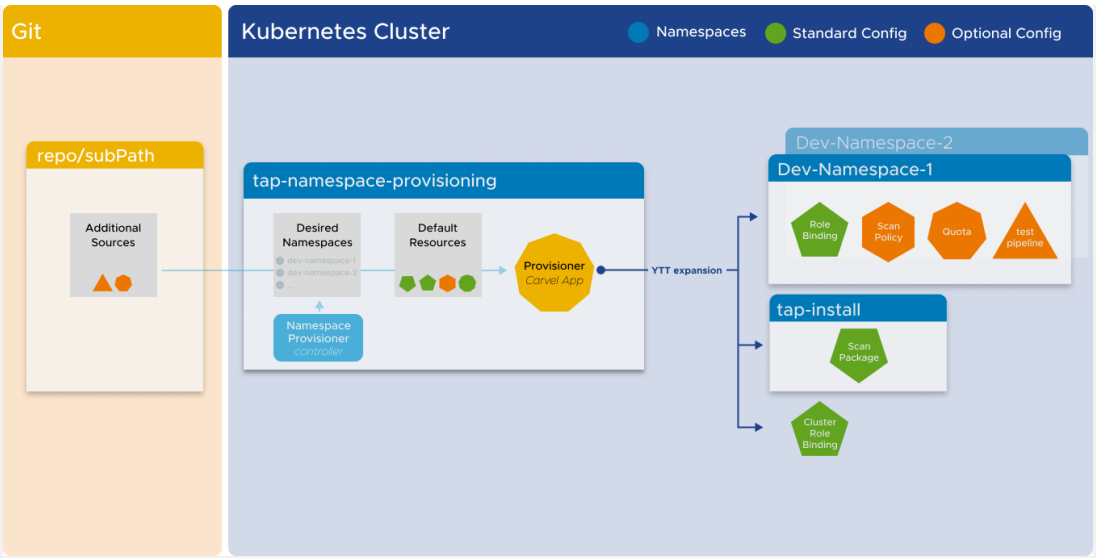
Namespace Provisioner enables platform operators to add additional customized namespace-scoped resources using GitOps to meet their organization's requirements and provides continuous reconciliation using the kapp-controller to maintain the desired state of the namespace-scoped resources.

Namespace Provisioner enables operators that are new to Kubernetes to automate the provisioning of multiple developer namespaces in a shared cluster. For organizations that have already adopted Kubernetes, Namespace Provisioner is also compatible with existing Kubernetes tooling.

Modes

Use Namespace Provisioner with one of the following modes:

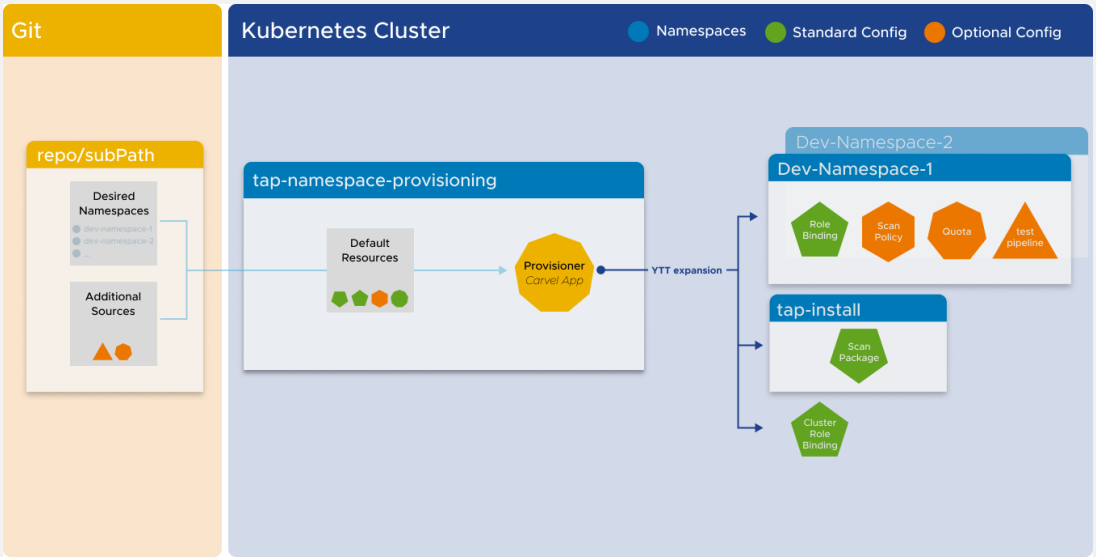
Controller mode
Controller mode has the following characteristics:



- The list of developer namespaces is managed by the Namespace Provisioner controller using a label selector `apps.tanzu.vmware.com/tap-ns=""`
- Namespace Provisioner creates default resources that are shipped Out of the Box in all managed namespaces.
- Namespace Provisioner creates additional Platform Operator templated resources stored in Git repository locations specified under the `additional_sources` section in Namespace Provisioner configuration. For more information, see Customize Installation of Namespace Provisioner.

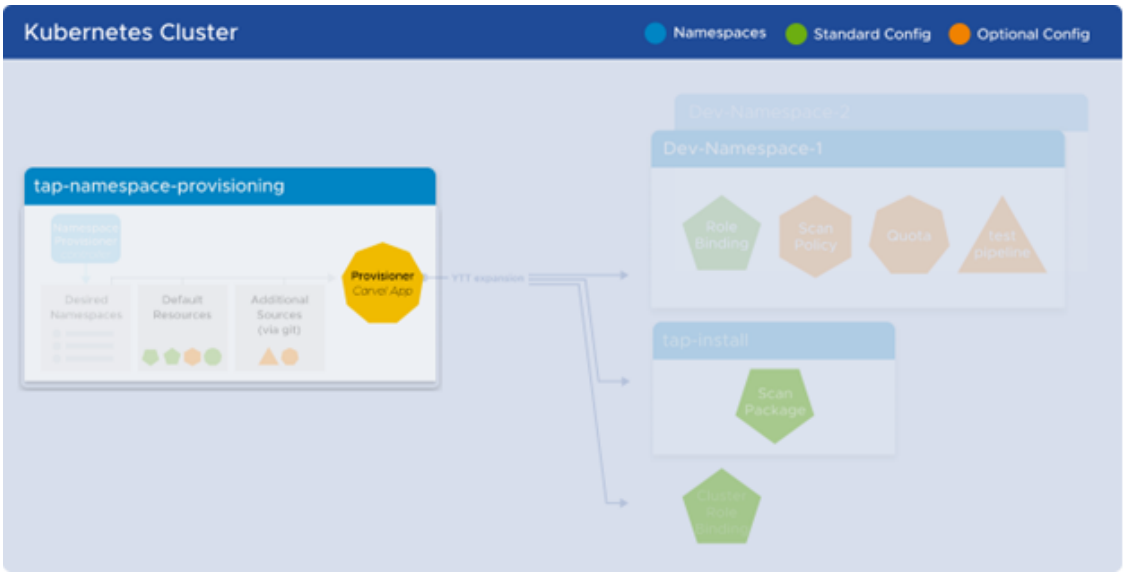
GitOps mode

Gitops mode has the following characteristics



- The list of developer namespaces is managed in a Git repository that is specified in the `gitops_install` section of the Namespace Provisioner configuration.
- Namespace Provisioner creates default resources that are shipped Out of the Box in all managed namespaces.
- Namespace Provisioner creates additional Platform Operator templated resources stored in Git repository locations specified under `additional_sources` in Namespace Provisioner configuration. For more information, see Customize Installation of Namespace Provisioner.

Provisioner Carvel application



Namespace Provisioner consists of a Carvel application called `provisioner` that facilitates the creation of resources in the managed developer namespaces. The `provisioner` application uses ytt to templatize a set of resources into installations in multiple namespaces.

Desired namespaces

The following section describes how the list of desired developer namespaces is managed in controller and GitOps modes.

Controller mode

In controller mode, the list of desired namespaces used by the `provisioner` application to create resources in, is maintained in the `desired-namespaces` ConfigMap. This ConfigMap is managed by the Namespace Provisioner controller and it provides a declarative way to indicate which namespaces should be populated with resources. The ConfigMap consists of a list of namespace objects, with a required `name` parameter, and optional additional parameters which are used as `data.values` for customizing defined resources.

For example,

```
---
apiVersion: v1
kind: ConfigMap
metadata:
    name: desired-namespaces
    namespace: tap-namespace-provisioning
    annotations:
        kapp.k14s.io/create-strategy: fallback-on-update
        namespace-provisioner.apps.tanzu.vmware.com/no-overwrite: "" #! This annotat
ion tells the provisioner app to not override this configMap as this is your desired
state.
data:
    namespaces.yaml: |
        #@data/values
        ---
        namespaces:
        - name: dev-ns1
        # additional parameters about dev-ns1 added via label/annotations or GitOps
        - name: dev-ns2
        # additional parameters about dev-ns1 added via label/annotations or GitOps
```

GitOps mode
In the GitOps mode, the list of desired namespaces used by the `provisioner` application to create resources in, is maintained in a Git repository as a ytt data values file as shown in this sample file. This file provides a declarative way to indicate which namespaces should be populated with resources. For more information, see the `Options if using GitOps` section in Customize Install.

Namespace Provisioner controller

The Namespace Provisioner controller (controller) is installed by default and manages the content contained in the `desired-namespaces` ConfigMap. The controller watches namespaces in the cluster and updates the `desired-namespaces` ConfigMap with a list of all namespaces that match the namespace label selector.The defalut namespace label selector is `apps.tanzu.vmware.com/tap-ns`. For more information, see Use a different label selector than default.

Get started with Namespace Provisioner

This topic provides a list of topics to help you get started with Namespace Provisioner.

Provision Developer Namespaces

Customize Installation of Namespace Provisioner

Setup for OOTB Supply Chains

Provision developer namespaces in Namespace Provisioner

This topic describes how to use Namespace Provisioner to provision developer namespaces in Tanzu Application Platform (commonly known as TAP).

Prerequisite

- The Namespace Provisioner package is installed and reconciled.
- The registry-credential secret referenced by the Supply chain components for pulling and pushing images is added to **tap-install** and exported to all namespaces.

Example secret creation, exported to all namespaces:

```
tanzu secret registry add registry-credentials --server REGISTRY-SERVER --username REG
ISTRY-USERNAME --password REGISTRY-PASSWORD --export-to-all-namespaces --yes --namespa
ce tap-install
```

Important

Namespace Provisioner creates a secret called registries-credentials in each managed namespace which is a placeholder secret filled indirectly by secretgen-controller with all the registry credentials exported for that managed namespace.

Manage a list of developer namespaces

There are two ways to manage the list of developer namespaces that are managed by Namespace Provisioner.

Using Namespace Provisioner Controller

Sample TAP values configuration:

```
namespace_provisioner:
  controller: true
```

The imperative way is to create the namespace using kubectl or using other means and label it using the default selector.

1. Create a namespace using kubectl or any other means

   ```
   kubectl create namespace YOUR-NEW-DEVELOPER-NAMESPACE
   ```

2. Label your new developer namespace with the default namespace_selector `apps.tanzu.vmware.com/tap-ns=""`.

   ```
   kubectl label namespaces YOUR-NEW-DEVELOPER-NAMESPACE apps.tanzu.vmware.com/t
   ap-ns=""
   ```

   - This label tells the Namespace Provisioner controller to add this namespace to the desired-namespaces ConfigMap.
   - By default, the label's value can be anything, including "".
   - If required, you can change the default label selector, see Customize Installation of Namespace Provisioner.

3. Run the following command to verify the default resources have been created in the namespace:

   ```
   kubectl get secrets,serviceaccount,rolebinding,pods,workload,configmap,limitr
   ange -n YOUR-NEW-DEVELOPER-NAMESPACE
   ```

   For example:

   ```
   NAME                              TYPE                                  DATA   AGE
   secret/app-tls-cert               kubernetes.io/tls                     3      19s
   secret/registries-credentials     kubernetes.io/dockerconfigjson        1      26s
   secret/scanner-secret-ref         kubernetes.io/dockerconfigjson        1      20s

   NAME                              SECRETS   AGE
   serviceaccount/default            1         4h7m
   serviceaccount/grype-scanner      2         20s

   NAME                                                                  ROLE
   AGE
   rolebinding.rbac.authorization.k8s.io/default-permit-deliverable   ClusterRol
   e/deliverable   26s
   rolebinding.rbac.authorization.k8s.io/default-permit-workload      ClusterRol
   e/workload      26s

   NAME                            DATA   AGE
   configmap/kube-root-ca.crt      1      38h

   NAME                               CREATED AT
   limitrange/dev-lr   2023-03-08T04:18:58Z
   ```

Using GitOps

The GitOps approach provides a fully declarative way to create developer namespaces managed by Namespace Provisioner.

Sample Tanzu Application Platform values configuration:

```
namespace_provisioner:
  controller: false
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

This GitOps configuration does the following things:

- `controller: false` - The Namespace Provisioner package does not install the controller. The list of namespaces is managed in a GitOps repository instead.
- The `gitops-install` directory specified as the `subPath` value includes two files:
    1. desired-namespace.yaml contains the list of developer namespaces in a ytt data.values format.
    2. namespaces.yaml contains a Kubernetes namespace object.

Note

If you have another tool like Tanzu Mission Control or some other process that is taking care of creating namespaces for you, and you don't want a Namespace Provisioner to create the namespaces, you can delete this file from your GitOps install repository.

Important The GitOps sample creates the following two namespaces: `dev` and `qa`. If these namespaces already exist in your cluster, remove them or rename the namespaces in your GitOps repository so they do not conflict with existing resources.

Run the following command to verify the default resources are created in the namespace:

```
kubectl get secrets,serviceaccount,rolebinding,pods,workload,configmap,limitrange -n
dev
```

For example:

```
NAME                              TYPE                                  DATA   AGE
secret/app-tls-cert               kubernetes.io/tls                     3      52s
secret/registries-credentials     kubernetes.io/dockerconfigjson        1      59s
secret/scanner-secret-ref         kubernetes.io/dockerconfigjson        1      53s

NAME                              SECRETS   AGE
serviceaccount/default            1         59s
serviceaccount/grype-scanner      2         53s

NAME                                                                ROLE
AGE
rolebinding.rbac.authorization.k8s.io/default-permit-deliverable    ClusterRole/deliv
erable   59s
rolebinding.rbac.authorization.k8s.io/default-permit-workload       ClusterRole/workl
oad      59s

NAME                           DATA   AGE
configmap/kube-root-ca.crt     1      59s

NAME                  CREATED AT
limitrange/dev-lr     2023-03-08T04:22:20Z
```

For more information, see the GitOps section of Customize Installation of Namespace Provisioner.

Enable additional users with Kubernetes RBAC

Namespace Provisioner does not support enabling additional users with Kubernetes RBAC. Support is planned for an upcoming release. Until Namespace Provisioner support is provided, follow the instructions in Enable additional users with Kubernetes RBAC.

Customize Namespace Provisioner installation

This topic tells you how to customize a standard installation of Namespace Provisioner in Tanzu Application Platform (commonly known as TAP).

Namespace Provisioner is packaged and distributed using a set of Carvel tools. The Namespace Provisioner package is installed as part of all the standard installation profiles except the View profile. For more information about installation profiles, see Installation profiles in Tanzu Application Platform.

The default set of resources provisioned in a namespace is based on a combination of the Tanzu Application Platform installation profile employed and the supply chain that is installed on the cluster. For a list of what resources are created for different profile and supply chain combinations, see the Default Resources mapping table.

To see the Namespace Provisioner Package Schema for all configurable values, run:

```
tanzu package available get namespace-provisioner.apps.tanzu.vmware.com/0.3.0 --values
-schema -n tap-install
```

Different package customization options are available depending on what method you use to manage the list of developer namespaces:

Options if using Controller
The following customization options are available if you are using the controller to manage the list of developer namespaces:

- Add additional resources to your namespaces from your GitOps repository
- Adjust sync period of Namespace Provisioner
- Import user defined secrets in YAML format as ytt data.values
- Use a different label selector than default
- Override default CPU and memory limits for controller pods
- Use AWS IAM roles
- Apply default parameters to all namespaces
- Customize the label and annotation prefixes that controller watches
- Import Overlay secrets

Add additional resources to your namespaces from your GitOps repository

- `additional_sources` is an array of Git repository locations that contain Platform Operator templated resources to create in the provisioned namespaces, in addition to the default resources. See the “fetch” section of the kapp controller App specification for the format. Only the Git type fetch is supported.

- `additional_sources[].git` can have a secretRef specified for providing authentication details for connecting to a private Git repository. For more information, see Git Authentication for Private repository. The following parameters are available:
    - `name`: name of the secret to be imported to use as valuesFrom in kapp.
    - `namespace`: namespace where the secret exists.
    - `create_export`: Boolean flag to control creation of a SecretExport resource in the namespace. The default value is false. If the secret is already exported, ensure that it is exported to the `tap-namespace-provisioning` namespace.
    - `path` must start with the prefix `_ytt_lib/`. Namespace Provisioner mounts all the additional sources as a ytt library so it can expand the manifests in the additional sources for all managed namespaces using the logic in the expansion template. The path after the `_ytt_lib` prefix can be any string value, and must be unique across all additional sources.

Important

Namespace Provisioner relies on kapp-controller for any tasks involving communication with external services, such as registries or Git repositories. When operating in air-gapped environments or other scenarios where external services are secured by a Custom CA certificate, you must configure kapp-controller with the CA certificate data to prevent X.509 certificate errors. For more information, see Deploy onto Cluster in the Cluster Essentials for VMware Tanzu documentation.

```
namespace_provisioner:
 controller: true
 additional_sources:
 - git:
     ref: origin/main
     subPath: ns-provisioner-samples/testing-scanning-supplychain
     url: https://github.com/vmware-tanzu/application-accelerator-samples.git
     # secretRef section is only needed if connecting to a Private Git repo
     secretRef:
       name: git-auth
       namespace: tap-install
       create_export: true
   path: _ytt_lib/testing-scanning-supplychain-setup
```

See Git Authentication for Private repository.

Adjust sync period of Namespace Provisioner

`sync_period` defines the wait time for the Namespace Provisioner to reconcile. `sync_period` is specified in time + unit format. The minimum `sync_period` allowed is 30 seconds. Namespace Provisioner sets the `sync_period` value to `30s` if a lesser value is specified in TAP values. If not specified, the value defaults to `1m0s`.

Sample TAP values configuration:

```
namespace_provisioner:
  sync_period: 2m0s
```

Import user defined secrets in YAML format as ytt data.values

`import_data_values_secrets` is an array of additional secrets in YAML format to import in the provisioner as data.values under the `data.values.imported` key. SecretImport for the secrets listed in the array is created in the `tap-namespace-provisioning` namespace by the Namespace

Provisioner package. Either, create SecretExport for the same secrets manually and export it to the `tap-namespace-provisioning` namespace, or let the Namespace Provisioner package create it. The following parameters are available:

- `name`: Name of the secret to be imported to use as valuesFrom in kapp.
- `namespace`: Namespace where the secret exists.
- `create_export`: Boolean flag to decide creation of a SecretExport resource in the namespace. The default value is `false`. If the secret is already exported, ensure that it is exported for the `tap-namespace-provisioning` namespace.

Note

The `stringData` key of the secret must have `.yaml` or `.yml` suffix.

Example secret:

```
# Format of the secret that is importable under data.values.imported
apiVersion: v1
kind: Secret
metadata:
  name: user-defined-secrets
type: Opaque
stringData:
  # Key needs to have .yaml or .yml at the end
  content.yaml: |
    key1: value1
    key2: value2
```

Sample TAP values configuration:

```
namespace_provisioner:
  controller: true
  import_data_values_secrets:
  - name: user-defined-secrets
    namespace: tap-install
    create_export: true
```

Use a different label selector than default

`namespace_selector` defines the label selector used by the controller to determine which namespaces should be added to the desired-namespaces ConfigMap.

Sample TAP values configuration:

```
namespace_provisioner:
  controller: true
  namespace_selector:
    matchExpressions:
    - key: apps.tanzu.vmware.com/tap-ns
      operator: Exists
```

Override default CPU and memory limits for controller pods

Use the `controller_resources` section in Namespace Provisioner configuration in TAP values to configure Namespace Provisioner Compute Resources controllers.

Set `controller_resources.resources.limits.cpu` and `controller_resources.resources.limits.memory` to configure the maximum CPU and memory available for the controller.

Similarly, set `controller_resources.resources.requests.cpu` and `controller_resources.resources.requests.memory` to configure the minimum CPU capacity and memory available for the controller.

Sample TAP values configuration:

```
namespace_provisioner:
  controller: true
  controller_resources:
    resources:
      limits:
        cpu: 500m
        memory: 100Mi
      requests:
        cpu: 100m
        memory: 20Mi
```

Use AWS IAM roles

If the TAP installation is on AWS with EKS, use the IAM Role from `aws_iam_role_arn` for the Kubernetes Service Account that is used by Workload and the Supply chain to create resources.

Sample TAP values configuration:

```
namespace_provisioner:
  controller: yes
  aws_iam_role_arn: "arn:aws:iam::123456789012:role/EKSIAMRole"
```

Apply default parameters to all namespaces

`default_parameters` is an array of parameters applied to all namespaces which can be used as ytt (`data.values.default_parameters`) for templating default and additional resources.

Sample TAP values configuration:

```
namespace_provisioner:
  controller: yes
  default_parameters:
    limits:
      default:
        cpu: 1.7
        memory: 1Gi
      defaultRequest:
        cpu: 100m
        memory: 1Gi
```

Customize the label and annotation prefixes that controller watches

`parameter_prefixes` is an array of label/annotation prefixes the controller looks for to add namespace specific parameters into the desired-namespaces ConfigMap which can be used as ytt data.values for templating default and additional resources. For example, the value `tap.tanzu.vmware.com` tells the Namespace Provisioner controller to look for the annotations/labels on a provisioned namespace that start with the prefix `tap.tanzu.vmware.com/` and use those as parameters.

Sample TAP values configuration:

```
namespace_provisioner:
  controller: yes
  parameter_prefixes:
  - tmc.cloud.vmware.com
  - tap.tanzu.vmware.com
```

Import overlay secrets

`overlay_secrets` is a list of secrets which contains Carvel ytt overlay definitions that are applied to the resources created by the Namespace Provisioner. The secrets are imported to `namespace-provisioner` namespace if it is in another namespace.

Note

The `stringData` key of the secret must have `.yaml` or `.yml` suffix.

Sample secret with overlay to be used:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: grype-package-overlay
  namespace: tap-install
  annotations:
    kapp.k14s.io/change-rule: "delete after deleting tap"
stringData:
  grype-package-overlay.yaml: |
    #@  load("@ytt:overlay", "overlay")
    #@
    #@  def matchGrypeScanners(index, left, right):
    #@    if left["apiVersion"] != "packaging.carvel.dev/v1alpha1" or left["kind"] !=
"PackageInstall":
    #@      return False
    #@    end
    #@    return "metadata" in left and "name" in left["metadata"] and left["metadat
a"]["name"].startswith("grype-scanner")
    #@  end

    #@overlay/match by=matchGrypeScanners, expects="0+"
    ---
    metadata:
      annotations:
        #@overlay/match expects="0+"
        ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: my-grype-overlay-secret
EOF
```

Sample TAP values configuration:

```
namespace_provisioner:
  controller: true
  overlay_secrets:
  - name: grype-package-overlay
    namespace: tap-install
    create_export: true
```

Options if using GitOps

The following customization options are available if you are using GitOps to manage the developer namespaces list:

- Use GitOps to manage developer namespaces list
- Add additional resources to your namespace from your GitOps repo
- Adjust sync period of Namespace Provisioner
- Import user defined secrets in YAML format as ytt data.values
- Use for AWS IAM roles

- Apply default parameters to all namespaces
- Import overlay secrets

Use GitOps to manage developer namespaces list

`gitops_install` is a Git repository configuration with the list of namespaces to be provisioned.

The `gitops_install` section must be used only when `controller: false` is set or else the Namespace Provisioner package fails to reconcile with the following error message: `controller: false when using 'gitops_install' in provided values`.

Files in the Git repository must have a `.yaml` or `.yml` extension.

The `gitops_install` section can have the following entries:

- `url`: the Git repository URL (mandatory)
- `subPath`: the Git repository subpath where the file is
- `ref`: the Git repository reference, the default is origin/main
- `secretRef`: if the repository needs authentication, the reference to the secret is set here
    - `name`: the name of the secret to be used for the repository authentication, see Git Authentication for Private repository.
    - `namespace`: the namespace where the secret is created. Namespace Provisioner creates a Carvel secretgen SecretImport from this given namespaces to Namespace Provisioner namespace.
    - `create_export`: Boolean flag to create a Carvel secretgen SecretExport from the given namespace to Namespace Provisioner namespace. The default value is `false`.

  Sample `gitops_install` repository file:

  Note

  The Carvel data header (#@data/values) is required in this file.

```
#@data/values
---
namespaces:
- name: dev
- name: qa
```

```
#@ load("@ytt:data", "data")
#! This loop will now loop over the namespace list in
#! in ns.yaml and will create those namespaces.
#@ for ns in data.values.namespaces:
---
apiVersion: v1
kind: Namespace
metadata:
  name: #@ ns.name
#@ end
```

This file in the sample repository creates the namespaces in the namespaces list so no manual intervention is required.

Sample TAP values configuration:

```
namespace_provisioner:
  controller: false
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Add additional resources to your namespace from your GitOps repo

- `additional_sources` is an array of locations of your Git repositories that contain Platform Operator templated resources to be created on the provisioned namespaces, in addition to the default resources.
- See the “fetch” section of the kapp controller App specification for the format. Only the Git type fetch is supported.
- `additional_sources[].git` can have secretRef specified for providing authentication details for connecting to a private Git repository. See Git Authentication for Private repository for more details. The following parameters are available:
    - `name`: name of the secret to be imported to use as valuesFrom in kapp.
    - `namespace`: namespace where the secret exists.
    - `create_export`: Boolean flag to decide creation of a SecretExport resource in the namespace. The default value is `false`. If the secret is already exported, ensure that it is exported for the `tap-namespace-provisioning` namespace.
- `path` must start with the prefix `_ytt_lib/`. Namespace Provisioner mounts all the additional sources as a ytt library so it can expand the manifests in the additional sources for all managed namespaces using the logic in the expansion template. The path after the `_ytt_lib` prefix can be any string value and must be unique across all additional sources.

Sample TAP values configuration:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
      # secretRef section is only needed if connecting to a Private Git repo
      secretRef:
        name: git-auth
        namespace: tap-install
        create_export: true
    path: _ytt_lib/testing-scanning-supplychain-setup
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

See Git Authentication for using a private Git repository guide.

Adjust sync period of Namespace Provisioner

`sync_period` defines the wait time for the Namespace Provisioner to reconcile. `sync_period` is specified in time + unit format. If a value less than 30 seconds is specified, it defaults to 30 seconds. If not specified, the value defaults to `1m0s`.

Sample TAP values configuration:

```
namespace_provisioner:
  sync_period: 1m0s
```

Import user defined secrets in YAML format as ytt data.values

`import_data_values_secrets` is an array of additional secrets in YAML format to import in the provisioner as data.values under the `data.values.imported` key. SecretImport for the secrets listed in the array are created in the `tap-namespace-provisioning` namespace by the Namespace Provisioner package. Either, create SecretExport for the same secrets manually and export it to `tap-namespace-provisioning` namespace or let the Namespace Provisioner package create it. Parameters include:

- `name`: Name of the secret to be imported to use as valuesFrom in kapp.
- `namespace`: Namespace where the secret exists.
- `create_export`: Boolean flag to decide creation of a SecretExport resource in the namespace. The default value is `false`. If the secret is already exported, ensure that it is exported for the `tap-namespace-provisioning` namespace.

Note

The `stringData` key of the secret must have `.yaml` or `.yml` suffix.

Example secret:

```
# Format of the secret that is importable under data.values.imported
apiVersion: v1
kind: Secret
metadata:
  name: user-defined-secrets
type: Opaque
stringData:
  # Key needs to have .yaml or .yml at the end
  content.yaml: |
    key1: value1
    key2: value2
```

Sample TAP values configuration:

```
namespace_provisioner:
  controller: false
  import_data_values_secrets:
  - name: user-defined-secrets
    namespace: tap-install
    create_export: true
```

Use for AWS IAM roles

If the TAP installation is on AWS with EKS, use the IAM Role from `aws_iam_role_arn` for the Kubernetes Service Account that is used by Workload and the Supply chain to create resources.

Sample TAP values configuration:

```
namespace_provisioner:
  controller: false
  aws_iam_role_arn: "arn:aws:iam::123456789012:role/EKSIAMRole"
```

Apply default parameters to all namespaces

`Default_parameters` is an array of parameters applied to all namespaces which can be used as ytt (`data.values.default_parameters`) for templating default and additional resources.

```
namespace_provisioner:
  controller: false
  default_parameters:
    limits:
      default:
        cpu: 1.7
        memory: 1Gi
      defaultRequest:
        cpu: 100m
        memory: 1Gi
```

Import Overlay secrets

`overlay_secrets` is a list of secrets which contains Carvel ytt overlay definitions that are applied to the resources created by the Namespace Provisioner. The secrets are imported to the `namespace-provisioner` namespace if it is in another namespace.

Note

The `stringData` key of the secret must have `.yaml` or `.yml` suffix.

Sample secret with overlay to be used

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: grype-package-overlay
  namespace: tap-install
  annotations:
    kapp.k14s.io/change-rule: "delete after deleting tap"
stringData:
  grype-package-overlay.yaml: |
    #@  load("@ytt:overlay", "overlay")
    #@
    #@  def matchGrypeScanners(index, left, right):
    #@    if left["apiVersion"] != "packaging.carvel.dev/v1alpha1" or left["kind"] !
= "PackageInstall":
    #@      return False
    #@    end
    #@    return "metadata" in left and "name" in left["metadata"] and left["metadat
a"]["name"].startswith("grype-scanner")
    #@  end

    #@overlay/match by=matchGrypeScanners, expects="0+"
    ---
    metadata:
      annotations:
        #@overlay/match expects="0+"
        ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: my-grype-overlay-secr
et
EOF
```

Sample TAP values configuration:

```
namespace_provisioner:
  controller: false
  overlay_secrets:
  - name: grype-package-overlay
```

```
    namespace: tap-install
    create_export: true
```

Set up Out of the Box Supply Chains in Namespace Provisioner

This topic tells you how to set up Namespace Provisioner to automate the creation of resources that are needed for workloads to run on Out of the Box Supply Chain Basic and Out of the Box Supply Chain with Testing.

Out of the Box Supply Chain Basic

To create a developer namespace, see Provision Developer Namespaces.

Namespace Provisioner creates a set of default resources in all managed namespaces which are sufficient to run a workload through the Out of the Box Supply Chain Basic.

Run the following Tanzu CLI command to create a workload in your developer namespace:

Using Tanzu CLI

Create workload using tanzu apps CLI command:

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
--app tanzu-java-web-app \
--namespace YOUR-NEW-DEVELOPER-NAMESPACE \
--tail \
--yes
```

Using workload yaml

Create a workload.yaml file with the details as below:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: YOUR-NEW-DEVELOPER-NAMESPACE
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Out of the Box Supply Chain with Testing

The Out of the Box Supply Chain with Testing adds the **source-tester** step in the supply chain which tests the source code pulled by the supply chain. For source code testing to work in the supply chain, a Tekton Pipeline must exist in the same namespace as the Workload so that, at the right moment, the Tekton PipelineRun object that is created to run the tests can reference the developer-provided Pipeline.

By default, the workload is matched to the corresponding pipeline to run using labels. Pipelines must have the label `apps.tanzu.vmware.com/pipeline: test` at a minimum. This provides a default match if no other labels are provided, but you can add additional labels for granularity. The pipeline expects two parameters:

- `source-url`, an HTTP address with a `.tar.gz` file containing all the source code to be tested
- `source-revision`, the revision of the commit or image reference (in case of `workload.spec.source.image` being set instead of `workload.spec.source.git`)

For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: tekton-pipeline-java
  labels:
    apps.tanzu.vmware.com/pipeline: test      # (!) required
spec:
  params:
    - name: source-url                        # (!) required
    - name: source-revision                   # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

Add a Java Tekton Pipeline to your developer namespace

To create a developer namespace, see the Provision Developer Namespaces.

Namespace Provisioner can automate the creation of a Tekton pipeline that is needed for the workload to run on an Out of the Box Supply Chain with Testing. You can create a sample pipeline in your GitOps repository and add your GitOps repository as an additional source in Namespace Provisioner configuration in TAP values. See Customize Installation of Namespace Provisioner.

Add the following configuration to your TAP values to add this sample java pipeline to your developer namespace:

Using Namespace Provisioner Controller
Sample TAP values configuration:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-supplychain
```

```
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-supplychain-setup
```

Using GitOps

Sample TAP values configuration:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-supplychain-setup
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

The sample pipeline resource has the following ytt logic which creates this pipeline only if the following conditions are met:

- `supply_chain` in your TAP values is either `testing` or `testing_scanning`
- `profile` in your TAP values is either `full, iterate`, or `build`.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in li
st)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('profil
e', ['full', 'iterate', 'build']):
```

After adding the additional source to your TAP values, you can see the `tekton-pipeline-java` created in your developer namespace. Run the following command to see if the pipeline is created correctly.

```
kubectl get pipeline.tekton.dev -n YOUR-NEW-DEVELOPER-NAMESPACE
```

Run the following Tanzu CLI command to create a workload in your developer namespace:

Using Tanzu CLI

Create workload using tanzu apps CLI command.

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
--app tanzu-java-web-app \
--label apps.tanzu.vmware.com/has-tests="true" \
--namespace YOUR-NEW-DEVELOPER-NAMESPACE \
--tail \
--yes
```

Using workload yaml

Create a workload.yaml file with the details as below.

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
```

```
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: YOUR-NEW-DEVELOPER-NAMESPACE
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Out of the Box Supply Chain with Testing and Scanning

The Out of the Box Supply Chain with Testing and Scanning adds the `source-tester`, `source-scanner`, and `image-scanner` steps in the supply chain which tests the source code pulled by the supply chain and scans for CVEs on the source and the image built by the supply chain. For these new testing and scanning steps to work, the following additional resources must exist in the same namespace as the workload.

- `Pipeline:` defines how to run the tests on the source code pulled by the supply chain and which image to use that has the tools to run those tests.
- `ScanTemplate`: defines how to run a scan, you can change how the scan is run, either for images or source code.
    - A ScanTemplate defines the PodTemplateSpec used by a Job to run a particular scan (image or source). When the supply chain initiates an ImageScan or SourceScan, they reference these templates which must be in the same namespace as the workload.
    - Although you can customize the templates, VMware recommends that you follow what is provided in the installation of the `grype.scanning.apps.tanzu.vmware.com` package. This is automatically created in all the namespaces managed by Namespace Provisioner. For more information, see About Source and Image Scans.
- `ScanPolicy`: define how to evaluate whether the artifacts scanned are compliant. For example, allowing one to be either very strict, or restrictive about particular vulnerabilities found.
    - When an ImageScan or a SourceScan is created to run a scan, they reference a policy, the policy name must match the following sample ScanPolicy.
    - See Writing Policy Templates.

Add a Java Tekton Pipeline & Grype Scan Policy to your developer namespace

To create a developer namespace, see Provision Developer Namespaces.

Namespace Provisioner can automate the creation of a Tekton pipeline and a ScanPolicy that is needed for the workload to run on an Out of the Box Supply Chain with Testing and Scanning. Create a sample Pipeline and a ScanPolicy in your GitOps repository and add your GitOps repository as an additional source in Namespace Provisioner configuration in TAP values. See Customize Installation of Namespace Provisioner for more details.

Add the following configuration to your TAP values to add the sample java pipeline and grype scan policyto your developer namespace:

Using Namespace Provisioner Controller

Sample TAP values configuration:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-scanning-supplychain-setup
```

Using GitOps

Sample TAP values configuration:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-scanning-supplychain-setup
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

The sample Pipeline resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your TAP values is either `testing` or `testing_scanning`
- `profile` in your TAP values is either `full, iterate`, or `build`.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in li
st)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('profil
e', ['full', 'iterate', 'build']):
```

The sample ScanPolicy resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your TAP values is `testing_scanning`
- `profile` in your TAP values is either `full` or `build`.

After adding the additional source to your TAP values, you can see the `tekton-pipeline-java and scan-policy` created in your developer namespace. Run the following command to see if the pipeline is created correctly.

```
kubectl get pipeline.tekton.dev,scanpolicies -n YOUR-NEW-DEVELOPER-NAMESPACE
```

Run the following Tanzu CLI command to create a workload in your developer namespace:

Using Tanzu CLI

Create workload using tanzu apps CLI command.

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
```

```
--git-branch main \
--type web \
--app tanzu-java-web-app \
--label apps.tanzu.vmware.com/has-tests="true" \
--namespace YOUR-NEW-DEVELOPER-NAMESPACE \
--tail \
--yes
```

Using workload yaml

Create a workload.yaml file with the details as below.

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: YOUR-NEW-DEVELOPER-NAMESPACE
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Namespace Provisioner use cases and examples

Review the following Namespace Provisioner uses cases:

Use multiple tekton pipelines and Scan policies in the same namespace in Namespace Provisioner

Add Tekton pipelines and scan policies using namespace parameters in Namespace Provisioner

Working with private Git repositories in Namespace Provisioner

Customize default resources in Namespace Provisioner

Install multiple scanners in the developer namespace in Namespace Provisioner

Use multiple Tekton pipelines and scan policies in the same namespace in Namespace Provisioner

This topic tells you how to use Namespace Provisioner to configure developer namespaces to include multiple Tekton pipelines and ScanPolices in Tanzu Application Platform (commonly known as TAP).

For information about, how to create a developer namespace, see Provision Developer Namespaces.

This sample GitOps location has a Java, Python and a Golang testing pipeline as well as a Strict and a Lax grype ScanPolicy.

Using Namespace Provisioner Controller

Add the following configuration to your TAP values to add multiple tekton pipelines and scan policies to your developer namespace:

```
namespace_provisioner:
  controller: true
```

```
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-polyglot
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-scanning-supplychain-polyglot-setup
```

Using GitOps

Add the following configuration to your TAP values to add multiple tekton pipelines and scan policies to your developer namespace:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-polyglot
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-scanning-supplychain-polyglot-setup
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

The sample Pipeline resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your TAP values is either `testing` or `testing_scanning`
- `profile` in your TAP values is either `full, iterate` or `build`.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in li
st)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('profil
e', ['full', 'iterate', 'build']):
```

All pipelines have an additional label `apps.tanzu.vmware.com/language` to differentiate between them.

The sample ScanPolicy resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your TAP values is `testing_scanning`
- `profile` in your TAP values is either `full` or `build`.

The strict ScanPolicy does not allow any workloads that have Critical and High vulnerabilities to pass through the supply chain whereas the lax ScanPolicy allows the workloads to pass regardless of CVEs detected. The allowed severity level is configured using the `notAllowedSeverities := []` part of the rego file section of ScanPolicy.

Caution

The lax ScanPolicy is just added for tutorial purposes but it is not advised to use such a policy in Production workloads.

After adding the additional source to your TAP values, you should be able to see the `tekton-pipeline-java, tekton-pipeline-golang, tekton-pipeline-python`, `scan-policy` and `lax-scan-`

`policy` created in your developer namespace. Run the following command to see if the pipelines are created correctly.

```
kubectl get pipeline.tekton.dev,scanpolicies -n YOUR-NEW-DEVELOPER-NAMESPACE
```

Run the following Tanzu CLI command to create a workload in your developer namespace:

Using Tanzu CLI

Create workload using tanzu apps CLI command

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
--app tanzu-java-web-app \
--label apps.tanzu.vmware.com/has-tests="true" \
--param-yaml testing_pipeline_matching_labels='{"apps.tanzu.vmware.com/language": "j
ava"}' \
--param scanning_source_policy="lax-scan-policy" \
--param scanning_image_policy="lax-scan-policy" \
--namespace YOUR-NEW-DEVELOPER-NAMESPACE \
--tail \
--yes
```

Using workload yaml

Create a workload.yaml file with the details as below.

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  generation: 1
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: YOUR-NEW-DEVELOPER-NAMESPACE
spec:
  params:
  - name: scanning_source_policy
    value: lax-scan-policy
  - name: scanning_image_policy
    value: lax-scan-policy
  - name: testing_pipeline_matching_labels
    value:
      apps.tanzu.vmware.com/language: java
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Note

`--param-yaml testing_pipeline_matching_labels` tells the supply chain to use the selector that matches the Java pipeline. To use the Python or Golang pipelines, use the selector that matches the language label in those resources.`--param`

`scanning_source_policy="lax-scan-policy"` tells the supply chain to use the lax ScanPolicy for the workload.

Add Tekton pipelines and scan policies using namespace parameters in Namespace Provisioner

This topic tells you how to use Namespace Provisioner to parameterize your additional resources and pass those parameters to namespaces in Tanzu Application Platform (commonly known as TAP).

Instead of creating all the pipelines in all provisioned namespaces, create a Tekton pipeline and ScanPolicy that is bespoke to namespaces that are running workloads using a specific language stack.

For information about, how to create a developer namespace, see Provision Developer Namespaces.

This use case looks at the pipelines and ScanPolicies in this sample GitOps location.

Using Namespace Provisioner Controller
When using the Namespace Provisioner controller, pass the parameters to a namespace via labels and annotations on the namespace. To enable this, set the `parameter_prefixes` in TAP configuration for Namespace Provisioner so the controller will look for labels/annotations starting with that prefix to populate parameters for a given namespace. See Controller section of Customize Installation of Namespace Provisioner guide for more information.

Add the following configuration to your TAP values to add parameterized tekton pipelines and scan policies to your developer namespace:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-parameterized
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-scanning-supplychain-parameterized-setup
  parameter_prefixes:
  - tap.tanzu.vmware.com
```

Note

We added `tap.tanzu.vmware.com` as a parameter_prefixes in Namespace Provisioner configuration. This tells the Namespace Provisioner controller to look for the annotations/labels on a provisioned namespace that start with the prefix `tap.tanzu.vmware.com/` and use those as parameters.

The sample pipelines have the following ytt logic which creates this pipeline only if

- `supply_chain` in your TAP values is either `testing` or `testing_scanning`
- `profile` in your TAP values is either`full, iterate` or `build`. `pipeline` parameter that matches the language for which the pipeline is for.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@   return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in
```

```
list)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('prof
ile', ['full', 'iterate', 'build']) and hasattr(data.values, 'pipeline') and data.va
lues.pipeline == 'java':
```

The sample ScanPolicy resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your TAP values is `testing_scanning`
- `profile` in your TAP values is either `full` or `build`.
- `scanpolicy`parameter matches either `strict` or `lax`

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in
list)
#@ end
#@ if/end in_list('supply_chain', ['testing_scanning']) and in_list('profile', ['ful
l', 'build']) and hasattr(data.values, 'scanpolicy') and data.values.scanpolicy ==
'lax':
```

Label your developer namespace using the `parameter_prefixes` with the parameter to be used in the `additional_sources` as follows:

```
kubectl label namespaces YOUR-NEW-DEVELOPER-NAMESPACE tap.tanzu.vmware.com/scanpolic
y=lax
```

```
kubectl label namespaces YOUR-NEW-DEVELOPER-NAMESPACE tap.tanzu.vmware.com/pipeline=
java
```

Using GitOps

When using GitOps, pass the parameters to a namespace by adding them to the data.values file located in our GitOps repo. Take a look at this sample file for an example.

Add the following configuration to your TAP values to add parameterized tekton pipelines and scan policies to your developer namespace:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-parameterized
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-scanning-supplychain-parameterized-setup
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install-with-params
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Note We added `gitops_install` with this sample GitOps location to create the namespaces and manage the desired namespaces from GitOps. See GitOps section of Customize Installation of Namespace Provisioner guide for more information.

Sample of `gitops_install` files:

```
#@data/values
---
namespaces:
- name: dev
  scanpolicy: lax
```

```
  pipeline: java
- name: qa
  scanpolicy: strict
  pipeline: java
```

```
#@ load("@ytt:data", "data")
#! This loop will now loop over the namespace list in
#! in ns.yaml and will create those namespaces.
#@ for ns in data.values.namespaces:
---
apiVersion: v1
kind: Namespace
metadata:
  name: #@ ns.name
#@ end
```

The sample pipelines have the following ytt logic which creates this pipeline only if

- `supply_chain` in your TAP values is either `testing` or `testing_scanning`
- `profile` in your TAP values is either`full, iterate` or `build`.
- `pipeline` parameter that matches the language for which the pipeline is for.

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in
list)
#@ end
#@ if/end in_list('supply_chain', ['testing', 'testing_scanning']) and in_list('prof
ile', ['full', 'iterate', 'build']) and hasattr(data.values, 'pipeline') and data.va
lues.pipeline == 'java':
```

The sample ScanPolicy resource have the following ytt logic which creates this pipeline only if

- `supply_chain` in your TAP values is `testing_scanning`
- `profile` in your TAP values is either `full` or `build`.
- `scanpolicy`parameter matches either `strict` or `lax`

```
#@ load("@ytt:data", "data")
#@ def in_list(key, list):
#@  return hasattr(data.values.tap_values, key) and (data.values.tap_values[key] in
list)
#@ end
#@ if/end in_list('supply_chain', ['testing_scanning']) and in_list('profile', ['ful
l', 'build']) and hasattr(data.values, 'scanpolicy') and data.values.scanpolicy ==
'lax':
```

Run the following Tanzu CLI command to create a workload in your developer namespace:

Using Tanzu CLI

Create workload using tanzu apps CLI command

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
--app tanzu-java-web-app \
--label apps.tanzu.vmware.com/has-tests="true" \
--namespace YOUR-NEW-DEVELOPER-NAMESPACE \
```

```
--tail \
--yes
```

Using workload yaml

Create a workload.yaml file with the details as below.

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: YOUR-NEW-DEVELOPER-NAMESPACE
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Run the following command to verify the resources have been created in the namespace: \

```
kubectl get secrets,serviceaccount,rolebinding,pods,workload,configmap,limitrange,pipeline,scanpolicies -n YOUR-NEW-DEVELOPER-NAMESPACE
```

Work with private Git repositories in Namespace Provisioner

This topic tells you how to configure Namespace Provisioner to use private Git repositories for storing GitOps based installation files, and platform operator templated resources that you want to create in your developer namespace in Tanzu Application Platform (commonly known as TAP).

Git Authentication for using a private Git repository

Namespaces Provisioner enables you to use private Git repositories for storing your GitOps based installation files as well as additional platform operator templated resources that you want to create in your developer namespace. Authentication is provided using a secret in `tap-namespace-provisioning` namespace, or an existing secret in another namespace referred to in the secretRef in the additional sources. For more information, see Customize Installation of Namespace Provisioner.

Create the Git Authentication secret in tap-namespace-provisioning namespace

The secrets for Git authentication allow the following keys: ssh-privatekey, ssh-knownhosts, username, and password.

Note

If ssh-knownhosts is not specified, Git does not perform strict host checking.
Important Namespace Provisioner relies on kapp-controller for any tasks involving communication with external services, such as registries or Git repositories. When operating in air-gapped environments or other scenarios where external services

are secured by a Custom CA certificate, you must configure kapp-controller with the CA certificate data to prevent X.509 certificate errors. For more information, see Deploy onto Cluster in the Cluster Essentials for VMware Tanzu documentation.

1. Create the Git secret.

Using HTTP(s) based Authentication
If you are using Username and Password for authentication:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: git-auth
  namespace: tap-namespace-provisioning
type: Opaque
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD
EOF
```

Using SSH based Authentication
If you are using SSH private key for authentication:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: git-auth
  namespace: tap-namespace-provisioning
type: Opaque
stringData:
  ssh-privatekey: |
      -----BEGIN OPENSSH PRIVATE KEY-----
      ..
      -----END OPENSSH PRIVATE KEY-----
EOF
```

2. Add the `secretRef` section to the `additional_sources` and the `gitops_install` section of the Namespace Provisioner configuration in your TAP values:

Using Namespace Provisioner Controller
Description

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: sources
      # This example URL is for SSH auth. Use https:// path if using HTTPS au
th
      url: git@github.com:private-repo-org/repo.git
      secretRef:
        name: git-auth
    path: _ytt_lib/my-additional-source
```

Using GitOps

Description

Caution There is a current limitation in kapp-controller which does not allow the users to re-use the same git secret multiple times. If you have multiple additional sources using private repo with the same credentials, you must create different secrets with the same authentication details for each of them.

In this example, the location where the list of namespaces resides is also a private repository. So you must create a secret named `git-auth-install` with the same authentication details.

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: tekton-pipelines
      # This example URL is for SSH auth. Use https:// path if using HTTPS au
th
      url: git@github.com:private-repo-org/repo.git
      secretRef:
        name: git-auth
    path: _ytt_lib/my-additional-source
  gitops_install:
    ref: origin/main
    subPath: gitops-install
    # This example URL is for SSH auth. Use https:// path if using HTTPS auth
    url: git@github.com:private-repo-org/repo.git
    secretRef:
      name: git-auth-install
```

Import from another namespace

If you already have a Git secret created in a namespace other than `tap-namespace-provisioning` namespace and you want to refer to that, the secretRef section should have the namespace mentioned along with the `create_export` flag. The default value for `create_export` is false as it assumes the Secret is already exported for tap-namespace-provisioning namespace, but allows the user to specify if they want the Namespace Provisioner to create a `Carvel SecretExport` for that secret.

The example refers to `git-auth` secret from `tap-install` in the secretRef section.

Using Namespace Provisioner Controller
Description

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: sources
      #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
      url: git@github.com:private-repo-org/repo.git
      secretRef:
          name: git-auth
          namespace: tap-install
          #! If this secret is already exported for this namespace, you can ignore t
he create_export key as it defaults to false
          create_export: true
    path: _ytt_lib/my-additional-source
```

Using GitOps

Description

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: tekton-pipelines
      #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
      url: git@github.com:private-repo-org/repo.git
      secretRef:
          name: git-auth
          namespace: tap-install
          #! If this secret is already exported for this namespace, you can ignore t
he create_export key as it defaults to false
          create_export: true
    path: _ytt_lib/my-additional-source
  gitops_install:
    ref: origin/main
    subPath: gitops-install
    #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
    url: git@github.com:private-repo-org/repo.git
    secretRef:
      name: git-auth-install
      namespace: tap-install
      #! If this secret is already exported for this namespace, you can ignore the c
reate_export key as it defaults to false
      create_export: true
```

After reconciling, Namespace Provisioner creates:

- SecretExport for the secret in the provided namespace (tap-install in the above example) to the Namespace Provisioner namespace.
- SecretImport for the secret in Namespace Provisioning namespace (tap-namespace-provisioning) so Carvel secretgen-controller can create the required secret for the Provisioner to connect to the Private Git Repository.

Git Authentication for Private Repository for Workloads and Supply chain

To either fetch or push source code from or to a repository that requires credentials, you must provide those through a Kubernetes secret object referenced by the intended Kubernetes object created for performing the action. The following sections provide details about how to appropriately set up Kubernetes secrets for carrying those credentials forward to the proper resources.

This section provides instructions on how to configure the `default` service account to work with private Git repositories for workloads and supply chain using Namespace Provisioner.

To configure the service account to work with private Git repositories, follow the steps below:

1. Create a secret in the `tap-install` namespace or any namespace of your preference, that contains the Git credentials in the YAML format.
    - `host`, `username` and `password` or `personal access token` values for HTTP based Git Authentication.
    - `ssh-privatekey`, `identity`, `identity_pub`, and `known_hosts` for SSH based Git Authentication.

Note

stringData key of the secret must have **.yaml** or **.yml** suffix at the end.

Using HTTP(s) based Authentication

If using Username and Password for authentication.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth
  namespace: tap-install
type: Opaque
stringData:
  content.yaml: |
    git:
      #! For HTTP Auth. Recommend using https:// for the git server.
      host: GIT-SERVER
      username: GIT-USERNAME
      password: GIT-PASSWORD
EOF
```

Using SSH based Authentication

If using SSH private key for authentication, create the git secret with authentication details as follows:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth
  namespace: tap-install
type: Opaque
stringData:
  content.yaml: |
    git:
      #! For SSH Auth
      ssh_privatekey: SSH-PRIVATE-KEY
      identity: SSH-PRIVATE-KEY
      identity_pub: SSH-PUBLIC-KEY
      known_hosts: GIT-SERVER-PUBLIC-KEYS
      host: GIT-SERVER
EOF
```

2. Create a scaffolding of a Git secret that needs to be added to the service account in the developer namespace in the GitOps repository. See the sample secret here. An example secret would look like the following. Instead of putting the actual username and password in the secret in the Git repository, put the reference to the values in the git-auth secret created in Step 1 by using the `data.values.imported` keys.

Using HTTP(s) based Authentication

If using Username and Password for authentication.

```
#@ load("@ytt:data", "data")
---
apiVersion: v1
kind: Secret
```

```
metadata:
  name: git
  annotations:
    tekton.dev/git-0: #@ data.values.imported.git.host
type: kubernetes.io/basic-auth
stringData:
  username: #@ data.values.imported.git.username
  password: #@ data.values.imported.git.token
```

Using SSH based Authentication

If using SSH private key for authentication:

```
#@ load("@ytt:data", "data")
---
apiVersion: v1
kind: Secret
metadata:
  name: git
  annotations:
    tekton.dev/git-0: #@ data.values.imported.git.host
type: kubernetes.io/ssh-auth
stringData:
  identity: #@ data.values.imported.git.identity
  identity.pub: #@ data.values.imported.git.identity_pub
  known_hosts: #@ data.values.imported.git.known_hosts
  ssh-privatekey: #@ data.values.imported.git.ssh_privatekey
```

3. Create a secret to specify an overlay to patch the default service account adding a reference to the secret **git**.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth-overlay
  namespace: tap-install
  annotations:
    kapp.k14s.io/change-rule: "delete after deleting tap"
stringData:
  workload-git-auth-overlay.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"apiVersion": "v1", "kind": "ServiceAcco
unt","metadata":{"name":"default"}}), expects="0+"
    ---
    secrets:
    #@overlay/append
    - name: git
EOF
```

4. Put all this together in Namespace Provisioner configuration in TAP values as follows:

Using Namespace Provisioner Controller

Add the following configuration to your TAP values

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://github.com/vmware-tanzu/application-accelerator-samples.gi
t
```

```
    path: _ytt_lib/credentials-setup
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  overlay_secrets:
  - name: workload-git-auth-overlay
    namespace: tap-install
    create_export: true
```

Using GitOps

Add the following configuration to your TAP values

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://github.com/vmware-tanzu/application-accelerator-samples.gi
t
    path: _ytt_lib/credentials-setup
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  overlay_secrets:
  - name: workload-git-auth-overlay
    namespace: tap-install
    create_export: true
```

5. First additional source points to the location where the templated git secret resides which is created in all developer namespaces.
6. Second additional source points to the overlay file which adds the git secret onto the default service account
7. Finally, import the newly created `workload-git-auth` secret into Namespace Provisioner to use in `data.values.imported` by adding the secret to the `import_data_values_secrets`.

Note

`create_export` is set to `true` in `import_data_values_secrets` meaning that a SecretExport is created for the `workload-git-auth` secret in the tap-install namespace automatically by Namespace Provisioner. After the changes are reconciled, the secret named **git** is in all provisioned namespaces and it is also added to the default service account of those namespaces.

Customize default resources in Namespace Provisioner

This topic tells you how to deactivate Grype in Namespace Provisioner and how to configure the `default` service account to work with private Git repositories in Tanzu Application Platform (commonly known as TAP).

Disable Grype install

Namespace Provisioner creates Grype scanner install as one of the default resources. If you choose to use another scanner for namespaces instead of Grype, you can disable the installation of the Out-of-the-box Grype scanner as follows:

1. Create an overlay secret as follows which removes the Grype scanner and the secret that is automatically created by Namespace Provisioner.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: disable-ootb-grype-overlay
  namespace: tap-install
  annotations:
    kapp.k14s.io/change-rule: "delete after deleting tap"
stringData:
  disable-ootb-grype-overlay.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@ def matchGrypeStuff(index, left, right):
    #@   if (left["apiVersion"] != "packaging.carvel.dev/v1alpha1" or left["kin
d"] != "PackageInstall") and (left["kind"] != "Secret"):
    #@     return False
    #@   end
    #@   return "metadata" in left and "name" in left["metadata"] and left["met
adata"]["name"].startswith("grype-scanner")
    #@ end

    #@overlay/match by=matchGrypeStuff, expects="0+"
    ---
EOF
```

2. Import this overlay secret onto Namespace Provisioner configuration so it gets applied to the resources created by Namespace Provisioner for all managed namespaces.

Using Namespace Provisioner Controller

Add the following configuration to your TAP values

```
namespace_provisioner:
  controller: true
  overlay_secrets:
  - name: disable-ootb-grype-overlay
    namespace: tap-install
    create_export: true
```

Using GitOps

Add the following configuration to your TAP values

```
namespace_provisioner:
  controller: false
  overlay_secrets:
  - name: disable-ootb-grype-overlay
    namespace: tap-install
    create_export: true
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Customize service accounts

This section provides instructions on how to configure the `default` service account to work with private Git repositories for workloads and supply chain using Namespace Provisioner.

To configure the service account to work with private Git repositories, follow the steps below:

1. Create a secret in the `tap-install` namespace (or any namespace of your preference) that contains the Git credentials in the YAML format.
    - `host`, `username` and `password` values for HTTP based Git Authentication.
    - `ssh-privatekey, identity, identity_pub`, and `known_hosts` for SSH based Git Authentication.

    Note

    stringData key of the secret must have **.yaml** or **.yml** suffix at the end.

    ```
    #! Example shows HTTP as well as SSH based authentication
    cat << EOF | kubectl apply -f -
    apiVersion: v1
    kind: Secret
    metadata:
      name: workload-git-auth
      namespace: tap-install
    type: Opaque
    stringData:
      content.yaml: |
        git:
          #! For HTTP Auth. Recommend using https:// for the git server.
          host: GIT-SERVER
          username: GIT-USERNAME
          token: GIT-PASSWORD
          #! For SSH Auth
          ssh_privatekey: SSH-PRIVATE-KEY
          identity: SSH-PRIVATE-KEY
          identity_pub: SSH-PUBLIC-KEY
          known_hosts: GIT-SERVER-PUBLIC-KEYS
    EOF
    ```

2. Create a scaffolding of a Git secret that needs to be added to the service account in your developer namespace in your GitOps repository. See the sample secret here. An example secret would look like the following. Instead of putting the actual username and password in the secret in your Git repository, put the reference to the values in the git-auth secret created in Step 1 by using the `data.values.imported` keys.

    ```
    #@ load("@ytt:data", "data")
    #@ load("@ytt:base64", "base64")
    ---
    apiVersion: v1
    kind: Secret
    metadata:
      name: git
      annotations:
        tekton.dev/git-0: #@ data.values.imported.git.host
    type: kubernetes.io/basic-auth
    stringData:
      username: #@ base64.encode(data.values.imported.git.username)
      password: #@ base64.encode(data.values.imported.git.token)
    ```

3. Create a secret to specify an overlay to patch the default service account adding reference to the secret git.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth-overlay
  namespace: tap-install
  annotations:
    kapp.k14s.io/change-rule: "delete after deleting tap"
stringData:
  workload-git-auth-overlay.yaml: |
    #@ load("@ytt:overlay", "overlay")
    #@overlay/match by=overlay.subset({"apiVersion": "v1", "kind": "ServiceAcco
unt","metadata":{"name":"default"}}), expects="0+"
    ---
    secrets:
    #@overlay/append
    - name: git
EOF
```

4. Put all this together in Namespace Provisioner configuration in TAP values as follows:

Using Namespace Provisioner Controller

Add the following configuration to your TAP values

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/credentials-setup
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  overlay_secrets:
  - name: workload-git-auth-overlay
    namespace: tap-install
    create_export: true
```

Using GitOps

Add the following configuration to your TAP values

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/credentials-setup
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
```

```
  overlay_secrets:
  - name: workload-git-auth-overlay
    namespace: tap-install
    create_export: true
```

- First additional source points to the location where our templated git secret resides which will be created in all developer namespaces.
- Second additional source points to the overlay file which will add the git secret onto the default service account
- Finally, import the newly created `workload-git-auth` secret into Namespace Provisioner to use in `data.values.imported` by adding the secret to the `import_data_values_secrets`.

Note

`create_export` is set to `true` in `import_data_values_secrets` meaning that a SecretExport will be created for the `workload-git-auth` secret in the tap-install namespace automatically by Namespace Provisioner. After the changes are reconciled, you will see the secret named git in all provisioned namespaces and also added to the default service account of those namespaces.

Customize Limit Range defaults

Namespace Provisioner creates LimitRange resource, see Default Resources in all namespaces managed by provisioner. Default values in LimitRange resource are as follows:

```
limits:
  default:
    cpu : 1500m
    memory : 1Gi
  defaultRequest:
    cpu : 100m
    memory : 1Gi
```

Update LimitRange defaults for all namespaces

Namespace Provisioner provides options for updating the values in LimitRange for all namespaces managed by the provisioner by specifying the `default_parameters` configuration in Namespace Provisioner TAP values as follows:

```
namespace_provisioner:
  default_parameters:
    # overwrite default limits set by the OOTB LimitRange for all namespaces
    limits:
      default:
        cpu: 1000m
        memory: 1Gi
      defaultRequest:
        cpu: 200m
        memory: 500Mi
```

Update LimitRange defaults for a specific namespace

If you wish to override the LimitRange for specific namespaces, you can do that via namespace parameters that can be applied as follows.

Using Namespace Provisioner Controller

User can annotate/label namespace using the default *parameter_prefix* `param.nsp.tap/` followed by the YAML path to cpu or memory limits as follows:

```
kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/limits.default.cpu=11
00m

kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/limits.default.memory
=2Gi

kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/limits.defaultReques
t.cpu=1500m

kubectl annotate ns YOUR-NEW-DEVELOPER-NAMESPACE param.nsp.tap/limits.defaultReques
t.memory=1Gi
```

- Controller will look at all the annotations/labels with prefix `param.nsp.tap/` and add the keys and its values in the desired-namespace configmaps as parameters for that namespace.
- Users can provide a custom prefix for the controller to look at if they do not want to use the default `param.nsp.tap` using parameter_prefixes configuration in Namespace Provisioner TAP values. See Controller Customization for more information on setting parameter_prefixes.

Note

Labels take precedence over annotations if the same key is provided in both.

Using GitOps

Add the following configuration to your TAP values to add parameterized limits to your developer namespace:

```
namespace_provisioner:
  controller: false
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install-with-params
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Note

We added `gitops_install` with this sample GitOps location to create the namespaces and manage the desired namespaces from GitOps. See GitOps section of Customize Installation of Namespace Provisioner guide for more information.

Sample of `gitops_install` files:

```
#@data/values
---
namespaces:
- name: dev
  limits:
    default:
      cpu: 1200m
      memory: 1.5Gi
```

```
    defaultRequest:
      cpu: 300m
      memory: 30Mi
- name: qa
```

```
#@ load("@ytt:data", "data")
#! This loop will now loop over the namespace list in
#! in ns.yaml and will create those namespaces.
#@ for ns in data.values.namespaces:
---
apiVersion: v1
kind: Namespace
metadata:
  name: #@ ns.name
#@ end
```

The Namespace Provisioner will create a LimitRange with default values for `qa` namespace and with the given values for `dev` namespace.

Install multiple scanners in the developer namespace in Namespace Provisioner

This topic tells you how to use Namespace Provisioner to automate multiple scanner installations in the developer namespace in Tanzu Application Platform (commonly known as TAP).

Grype scanner is installed by default in all namespaces managed by Namespace Provisioner.

The following step describe how to install Snyk scanner and Grype in the developer namespace and use both together in the supply chain. Grype is used for Source scans and Snyk is used for Image scans.

For information about, how to create a developer namespace, see Provision Developer Namespaces.

1. Create a secret in the `tap-install` namespace or any namespace of your preference that contains the Snyk token in the YAML format **(must have .yaml or .yml in the key)** as shown below:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: v1
   kind: Secret
   metadata:
     name: scanner-auth
     namespace: tap-install
   type: Opaque
   stringData:
     content.yaml: |
       scanners:
         snyk_api_token: "" # Paste your snyk API token here
   EOF
   ```

2. Add the following configuration to your TAP values to create the supply-chain and scanners:

Using Namespace Provisioner Controller
Add the following configuration to your TAP values

```
namespace_provisioner:
  controller: true
  additional_sources:
```

```
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-multiple-scanners
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-scanning-supplychain-multiple-scanners-setup
  import_data_values_secrets:
  - name: scanner-auth
    namespace: tap-install
    create_export: true
```

Using GitOps

Add the following configuration to your TAP values

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/testing-scanning-supplychain-multiple-scanners
      url: https://github.com/vmware-tanzu/application-accelerator-samples.git
    path: _ytt_lib/testing-scanning-supplychain-multiple-scanners-setup
  import_data_values_secrets:
  - name: scanner-auth
    namespace: tap-install
    create_export: true
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://github.com/vmware-tanzu/application-accelerator-samples.git
```

Additional source points to the location of the sample GitOps repo where we have the following custom resources:

- `tekton-pipeline-java.yaml` - For creating a Tekton pipeline for running tests on our Java workload
- `scanpolicy-grype.yaml and scanpolicy-snyk.yaml`- For creating a Scan policies to be used by Grype and Snyk scanners
- `snyk-token-secret.yaml` -is a Snyk token secret that needs to be created in our developer namespace. Instead of putting the actual snyk token in the secret in our Git repository, we will put the reference to the values in the scanner-auth secret created in Step 1 by using the `data.values.imported` keys.
- `snyk-scanner-install.yaml` - contains the PackageInstall for installing the Snyk package for our developer namespace. One particular thing to note on this file is that we have mentioned the namespace: tap-install in the PackageInstall resource. This signals the Namespace Provisioner to create a PackageInstall resource for all provisioned namespaces in the same namespace (in our case tap-install) and add`-{namespace}` as the suffix in the name to avoid name collisions.

Our setup is complete. Run the following Tanzu CLI command to apply a workload in your developer namespace that uses Grype for source scan and Snyk for Image scan:

Using Tanzu CLI

Create workload using tanzu apps CLI command

```
tanzu apps workload apply tanzu-java-web-app \
--git-repo https://github.com/sample-accelerators/tanzu-java-web-app \
--git-branch main \
--type web \
```

```
--app tanzu-java-web-app \
--label apps.tanzu.vmware.com/has-tests="true" \
--param scanning_image_policy=snyk-scan-policy \
--param scanning_image_template=snyk-private-image-scan-template \
--namespace YOUR-NEW-DEVELOPER-NAMESPACE \
--tail \
--yes
```

Using workload yaml

Create a workload.yaml file with the details as below.

```
---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  labels:
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/has-tests: "true"
    apps.tanzu.vmware.com/workload-type: web
  name: tanzu-java-web-app
  namespace: YOUR-NEW-DEVELOPER-NAMESPACE
spec:
  params:
  - name: scanning_image_policy
    value: snyk-scan-policy
  - name: scanning_image_template
    value: snyk-private-image-scan-template
  source:
    git:
      ref:
        branch: main
      url: https://github.com/sample-accelerators/tanzu-java-web-app
```

Work with Git repositories in air-gapped environments with Namespace Provisioner

This topic provides instructions for configuring Namespace Provisioner to use air-gapped Git repositories. This allows you to store GitOps-based installation files and platform operator-templated resources intended for creation in your developer namespace in Tanzu Application Platform (TAP).

Git authentication

Authentication is established through a secret in the `tap-namespace-provisioning` namespace or an existing secret in another namespace referenced in the `secretRef` in `additional_sources`. For more details, refer to Customize Installation of Namespace Provisioner.

Create the Git authentication secret in `tap-namespace-provisioning` namespace

The Git authentication secrets support the following keys: `ssh-privatekey`, `ssh-knownhosts`, `username`, and `password`. If `ssh-knownhosts` is not specified, Git does not perform strict host checking.

Important

In air-gapped environments or other scenarios where external services are secured by a Custom CA certificate, configure kapp-controller with the CA certificate data to prevent X.509 certificate errors. For detailed instructions, refer to Deploy onto Cluster in the Cluster Essentials for VMware Tanzu documentation.

1. Create the Git secret:

Using HTTP(s) based authentication
If you are using user name and password for authentication:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: git-auth
  namespace: tap-namespace-provisioning
type: Opaque
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD
EOF
```

Using SSH based Authentication
If you are using SSH private key for authentication:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: git-auth
  namespace: tap-namespace-provisioning
type: Opaque
stringData:
  ssh-privatekey: |
      -----BEGIN OPENSSH PRIVATE KEY-----
      ...
      -----END OPENSSH PRIVATE KEY-----
EOF
```

2. Add the `secretRef` section to the `additional_sources` and the `gitops_install` section of your `tap-values.yaml` file:

Using Namespace Provisioner Controller
Description

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: sources
      # This example URL is for SSH auth. Use https:// path if using HTTPS au
th
      url: git@git-airgap-server:private-repo-org/repo.git
      secretRef:
        name: git-auth
```

Using GitOps

Description

Caution In kapp-controller v0.46.0 and earlier, there is a limitation that prevents the reuse of the same Git secret multiple times. If you have multiple additional sources using repositories with identical credentials, you must create distinct secrets, each with the same authentication details.

In this example, the list of namespaces resides in a repository. Therefore, you must create a secret named `git-auth-install` with the same authentication details for this location.

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: tekton-pipelines
      # This example URL is for SSH auth. Use https:// path if using HTTPS au
th
      url: git@git-airgap-server:private-repo-org/repo.git
      secretRef:
        name: git-auth
  gitops_install:
    ref: origin/main
    subPath: gitops-install
    # This example URL is for SSH auth. Use https:// path if using HTTPS auth
    url: git@git-airgap-server:private-repo-org/repo.git
    secretRef:
      name: git-auth-install
```

Import from another namespace

If you already have a Git secret created in a namespace other than the `tap-namespace-provisioning` namespace and you want to refer to it, the `secretRef` section must include the namespace and the `create_export` flag. The default value for `create_export` is `false`, assuming the secret is already exported for the `tap-namespace-provisioning` namespace. However, you can specify if you want Namespace Provisioner to create a Carvel `SecretExport` for that secret.

In this example, the `secretRef` section refers to the `git-auth` secret from the `tap-install` namespace.

Using Namespace Provisioner Controller

Description

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: sources
      #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
      url: git@git-airgap-server:private-repo-org/repo.git
      secretRef:
          name: git-auth
          namespace: tap-install
          #! If this secret is already exported for this namespace, you can ignore t
he create_export key as it defaults to false
          create_export: true
```

Using GitOps

Description

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: tekton-pipelines
      #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
      url: git@git-airgap-server:private-repo-org/repo.git
      secretRef:
          name: git-auth
          namespace: tap-install
          #! If this secret is already exported for this namespace, you can ignore t
he create_export key as it defaults to false
          create_export: true
  gitops_install:
    ref: origin/main
    subPath: gitops-install
    #! This example URL is for SSH auth. Use https:// path if using HTTPS auth
    url: git@git-airgap-server:private-repo-org/repo.git
    secretRef:
      name: git-auth-install
      namespace: tap-install
      #! If this secret is already exported for this namespace, you can ignore the c
reate_export key as it defaults to false
      create_export: true
```

After reconciliation, Namespace Provisioner creates:

- SecretExport for the secret in the provided namespace, exporting it to the Namespace Provisioner namespace, for example, `tap-install`
- SecretImport for the secret in the `tap-namespace-provisioning` namespace. This enables Carvel secretgen-controller to create the required secret, allowing Namespace Provisioner to connect to the Git repository.

Git authentication for workloads and supply chain

When fetching or pushing source code to a repository that requires credentials, it's essential to provide those credentials through a Kubernetes secret object referenced by the corresponding Kubernetes object created for the action. The following sections describe setting up Kubernetes secrets to securely pass these credentials to the relevant resources. This procedure provides the steps to configure the `default` service account to interact with Git repositories for workloads and supply chain using Namespace Provisioner.

Set up the service account to interact with Git repositories:

1. Create a secret in the `tap-install` namespace or any preferred namespace, containing Git credentials in YAML format.
    - `host`, `username`, `caFile` and `password` or `personal access token` values for HTTP-based Git authentication.
    - `ssh-privatekey`, `identity`, `identity_pub`, and `known_hosts` for SSH-based Git authentication.

Note

The `stringData` key of the secret must end with either the `.yaml` or `.yml` suffix.

Using HTTP(s) based authentication

If using user name and password for authentication.

In this configuration for an air-gapped environment, the Git repository server has a custom certificate of authority that cannot be verified against public issuers, so you must provide the `caFile` content to log in against it.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth
  namespace: tap-install
type: Opaque
stringData:
  content.yaml: |
    git:
      #! For HTTP Auth. Recommend using https:// for the git server.
      host: GIT-SERVER
      username: GIT-USERNAME
      password: GIT-PASSWORD
      caFile: |
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
EOF
```

Using SSH based authentication

To use an SSH private key for authentication, create the Git secret with the authentication details as follows:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: workload-git-auth
  namespace: tap-install
type: Opaque
stringData:
  content.yaml: |
    git:
      host: GIT-SERVER
      #! For SSH Auth
      ssh_privatekey: SSH-PRIVATE-KEY
      identity: SSH-PRIVATE-KEY
      identity_pub: SSH-PUBLIC-KEY
      known_hosts: GIT-SERVER-PUBLIC-KEYS
EOF
```

2. To create a secret to be added to the service account in the developer namespace in the GitOps repository, use this example for HTTP-based or this example for `setings.xml`-based, or follow the example below.

   Rather than directly including the actual user name and password in the Git repository secret, use the `data.values.imported` keys to add references to the values from the `git-auth` secret created in the previous step.

This secret represents the Git secret that is created by the Namespace Provisioner in each managed namespace. It must be included in your Git repository linked in the `additional_sources` section of `tap-values.yaml` mentioned in the next step.

Using HTTP(s) based authentication

If using user name and password for authentication.

In this configuration for an air-gapped environment, the Git repository server has a custom certificate of authority that cannot be verified against public issuers, so you must provide the `caFile` content to log in against it.

```
#@ load("@ytt:data", "data")
---
apiVersion: v1
kind: Secret
metadata:
 name: git
 annotations:
   tekton.dev/git-0: #@ data.values.imported.git.host
type: kubernetes.io/basic-auth
stringData:
 username: #@ data.values.imported.git.username
 password: #@ data.values.imported.git.token
 caFile: #@ data.values.imported.git.caFile
```

Using settings.xml based authentication for Java applications

If using user name and password for authentication.

```
#@ load("@ytt:data", "data")

apiVersion: v1
kind: Secret
metadata:
 name: settings-xml
type: service.binding/maven
stringData:
 type: maven
 provider: sample
 #@yaml/text-templated-strings
 settings.xml: |
   <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="htt
p://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 https://mave
n.apache.org/xsd/settings-1.0.0.xsd">
       <mirrors>
           <mirror>
               <id>reposilite</id>
               <name>Accelerator samples</name>
               <url>(@= data.values.imported.git.host @)/vmware-tanzu/applica
tion-accelerator-samples</url>
               <mirrorOf>*</mirrorOf>
           </mirror>
       </mirrors>
       <servers>
           <server>
               <id>reposilite</id>
               <username>(@= data.values.imported.git.username @)</username>
               <password>(@= data.values.imported.git.password @)</password>
           </server>
       </servers>
   </settings>
```

Using SSH based authentication

If using SSH private key for authentication:

```
#@ load("@ytt:data", "data")
---
apiVersion: v1
kind: Secret
metadata:
  name: git
  annotations:
    tekton.dev/git-0: #@ data.values.imported.git.host
type: kubernetes.io/basic-auth
stringData:
  identity: #@ data.values.imported.git.identity
  identity.pub: #@ data.values.imported.git.identity_pub
  known_hosts: #@ data.values.imported.git.known_hosts
  ssh-privatekey: #@ data.values.imported.git.ssh_privatekey
```

3. Combine this `tap-values.yaml`:

Using Namespace Provisioner Controller

Add the following configuration to `tap-values.yaml`:

```
namespace_provisioner:
  controller: true
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://git-airgap-server/application-accelerator-samples.git
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  default_parameters:
    supply_chain_service_account:
      secrets:
      - git
```

Where `https://git-airgap-server/application-accelerator-samples.git` is a fork of the application-accelerator-samples repository.

Using GitOps

Add the following configuration to `tap-values.yaml`:

```
namespace_provisioner:
  controller: false
  additional_sources:
  - git:
      ref: origin/main
      subPath: ns-provisioner-samples/credentials
      url: https://git-airgap-server/vmware-tanzu/application-accelerator-sam
ples.git
  gitops_install:
    ref: origin/main
    subPath: ns-provisioner-samples/gitops-install
    url: https://git-airgap-server/vmware-tanzu/application-accelerator-sampl
es.git
  import_data_values_secrets:
  - name: workload-git-auth
    namespace: tap-install
    create_export: true
  default_parameters:
```

```
    supply_chain_service_account:
      secrets:
      - git
```

Where `https://git-airgap-server/application-accelerator-samples.git` is a fork of the application-accelerator-samples repository.

- `additional sources` points to the location where the templated Git secret resides, which is created in all developer namespaces.
- Import the newly created `workload-git-auth` secret into Namespace Provisioner to use in `data.values.imported` by adding the secret to `import_data_values_secrets`.
- Add the secret to be included in the ServiceAccount in the `default_parameters`. For more information, see Customize service accounts.

Note

`create_export` is set to `true` in `import_data_values_secrets`. As a result, a SecretExport is automatically created for the `workload-git-auth` secret in the `tap-install` namespace by Namespace Provisioner. After the changes are reconciled, the secret named **git** is present in all provisioned namespaces and is also added to the default service account of those namespaces.

4. In your `tap-values.yaml` file, in the `ootb_supply_chain_*.gitops.ssh_secret` section, specify the name of the Git secret containing the credentials. This is necessary for the supply chain to include the `secretRef` when creating the Flux `GitRepository` resource. For example:

```
ootb_supply_chain_testing_scanning:
  gitops:
    ssh_secret: git  # Replace with the actual name of your Git secret for the
workload, if different
```

By providing this configuration, the supply chain associates the created `GitRepository` resource with the specified Git secret managed by Namespace Provisioner.

5. Create the workload:

Using HTTP/HTTPS or SSH-based

If using user name and password for authentication.

```
tanzu apps workload apply APP-NAME \
--git-repo GIT-REPO \
--git-branch BRANCH \
--type web \
--app APP-NAME \
--label apps.tanzu.vmware.com/has-tests="true" \
--namespace DEV-NAMESPACE \
--tail \
--yes
```

Using settings.xml based authentication for Java applications

If using user name and password for authentication.

```
tanzu apps workload apply APP-NAME \
--git-repo GIT_REPO \
--git-branch BRANCH \
```

```
--type web \
--app APP-NAME \
--label apps.tanzu.vmware.com/has-tests="true" \
--namespace DEV-NAMESPACE \
--param-yaml buildServiceBindings='[{"name": "settings-xml", "kind": "Secre
t"}]'
--tail \
--yes
```

Troubleshoot Namespace Provisioner

This topic tells you how to troubleshoot Namespace Provisioner in Tanzu Application Platform (commonly known as TAP).

Air-gapped installation

Namespace Provisioner relies on kapp-controller for any tasks involving communication with external services, such as registries or Git repositories. When operating in air-gapped environments or other scenarios where external services are secured by a Custom CA certificate, you must configure kapp-controller with the CA certificate data to prevent X.509 certificate errors. For more information, see Deploy onto Cluster in the Cluster Essentials for VMware Tanzu documentation.

View controller logs

To get the logs when using the controller workflow, run the following kubectl command:

```
kubectl -n tap-namespace-provisioning logs deployments/controller-manager
```

Use `-f` to follow the log output.

Provisioner application error

After the Namespace Provisioner is installed in the Tanzu Application Platform cluster, the main resource to check is the provisioner Carvel Application in the `tap-namespace-provisioning` namespace. To check for the status of the Application, run the following kubectl command:

```
kubectl -n tap-namespace-provisioning get app/provisioner --template={{.status.usefulE
rrorMessage}}
```

Common errors

You might encounter one of the following errors:

Namespace selector malformed

When using the controller and customizing the `namespace_selector` from `tap_values.yaml`, the match expression must be compliant with the Kubernetes label selector. If it is not compliant, the Namespace Provisioner controller fails and log an error message in the controller logs.

For example, if the configured `namespace_selector` is as follows:

```
namespace_provisioner:
  controller: true
  namespace_selector:
    matchExpressions:
```

```
    - key: apps.tanzu.vmware.com/tap-ns
      operator: exists
```

This is not compliant as the operator must be `Exist` instead of `exists`. When labeling the namespace `dev` with `apps.tanzu.vmware.com/tap-ns`, the controller produces an error message similar to the following, (followed by some reconciliation messages)

```
{"level":"error","ts":"2022-12-14T15:41:44.639402794Z","logger":".0.1.NamespaceSelecto
rReconciler","msg":"unable to sync","controller":"namespace","controllerGroup":"","con
trollerKind":"Namespace","Namespace":{"name":"dev"},"namespace":"","name":"dev","recon
cileID":"26395d34-418b-446d-9b5e-a4a73cc657ed","resourceType":"/v1, Kind=Namespace","e
rror":"\"exists\" is not a valid pod selector operator","stacktrace":"..."}
```

Debugging ytt templating errors in additional sources

When working with ytt, templating errors in the additional sources in your GitOps repository can cause the Provisioner Carvel application to go into `Reconcile Failed` state. To debug the Application, run the following command:

```
kubectl -n tap-namespace-provisioning get app/provisioner --template={{.status.usefulE
rrorMessage}}
```

Sample Error message from Application when there is a ytt templating error:

```
ytt: Error:
- library.eval: Evaluating library 'witherror':
    in <toplevel>
      template.yaml:155 | #@          instances = overlay.apply(instance.eval(), custo
mize())
    reason:
     - struct has no .gl_secret_user field or method
         in <toplevel>
           _ytt_lib/witherror/secrets.yaml:12 |   username: #@ data.values.gl_secret_u
ser
```

Unable to delete namespace

When a user tries to delete a namespace that was managed by Namespace Provisioner, it gets stucks in the `Terminating` status.

Possible Cause 1: When a provisioned namespace that has a Cartographer Workload in it is deleted, the namespace will likely remain in the Terminating state because some resources can not be deleted. One of the causes of this behavior is that the Cartographer Workload using the Out of the Box supply chains and delivery creates a Carvel Kapp App for the workload that references the ServiceAccount in the namespace. Deleting the namespace deletes the Service Account that Kapp relies on before the App itself is deleted. As a result, the Carvel App blocks the namespace termination while waiting for the ServiceAccount to exist with a `finalizer` (`finalizers.kapp-ctrl.k14s.io/delete`) message.

Solution: Remove the Kapp App finalizer in the Kapp App

Possible Cause 2: When a user tries to delete a namespace that was previously managed by the Namespace Provisioner controller, and the namespace was not cleaned up before disabling the controller, it gets stuck in the `Terminating` state. This happens because the Namespace Provisioner controller adds a `finalizer` to the namespaces (`namespace-provisioner.apps.tanzu.vmware.com/finalizer`) it manages, and is no longer there to clean up that finalizer as it was disabled by the user.

Solution: Remove manually the finalizer in the namespace

Namespace Provisioner reference

This section tells you about the resources that are templated in the default-resources secret for each installation profile and supply chain value combination.

If you are using a GitOps repository to manage the list of namespaces, all the namespaces in the list must exist in the cluster. The provisioner application fails to reconcile if the namespaces do not exist on the cluster.

If you switch from controller mode to GitOps mode, you must manually remove the finalizer on all the namespaces previously managed by the controller. For more information on using controller versus GitOps, see Modes.

To use different private repositories, the secret used for each entry (gitops_install, additional_sources) must be a unique name. Re-using the same secret is not supported due to a limitation in kapp-controller.

Default resources

Namespace Provisioner is installed as part of the standard installation profiles. The default set of resources provisioned in a namespace is based on a combination of the installation profile employed and the supply chain that is installed on the cluster. For more information about installation profiles, see Installation profiles in Tanzu Application Platform

The following table shows the list of resources that are templated in the default-resources secret for an installation profile and supply chain value combination:

Namespace	Kind	Name	supply_chain	Install Profile	Reconcile
tap-install	PackageInstall	grype-scanner-{ns}	testing_scanning	full, build	Yes
tap-install	Secret	grype-scanner-{ns}	testing_scanning	full, build	Yes
Developer Namespace	Secret	registries-credentials	n/a	full, iterate, build, run	Yes
Developer Namespace	ServiceAccount	From: ootb_supply_chain_{supply_chain}.service_account (default: “default”)	n/a	full, iterate, build, run	No
Developer Namespace	ServiceAccount	From: ootb_delivery_basic.service_account (default: “default”)	n/a	full, iterate, run	No
Developer Namespace	RoleBinding	default-permit-deliverable	n/a	full, iterate, run	Yes
Developer Namespace	RoleBinding	default-permit-workload	n/a	full, iterate, build	Yes
Developer Namespace	LimitRange	{namespace}-lr	n/a	full, iterate, build	Yes

For installing additional resources for OOTB Supply Chain with Testing and Scanning, see Supply Chain Security Tools - Scan.

Overview of Service Bindings

This topic tells you about using Service Bindings in Tanzu Application Platform (commonly know as TAP).

Supported service binding specifications

Service Bindings packages the Service Binding for Kubernetes open source project.

It implements the Service Binding Specification for Kubernetes v1.0.

This implementation provides support for:

- Provisioned Service
- Workload Projection
- Service Binding
- Direct Secret Reference
- Role-Based Access Control (RBAC)

The following are not supported:

- Workload Resource Mapping
- Extensions including:
    - Binding Secret Generation Strategies

Overview of Service Bindings

This topic tells you about using Service Bindings in Tanzu Application Platform (commonly know as TAP).

Supported service binding specifications

Service Bindings packages the Service Binding for Kubernetes open source project.

It implements the Service Binding Specification for Kubernetes v1.0.

This implementation provides support for:

- Provisioned Service
- Workload Projection
- Service Binding
- Direct Secret Reference
- Role-Based Access Control (RBAC)

The following are not supported:

- Workload Resource Mapping
- Extensions including:
    - Binding Secret Generation Strategies

Install Service Bindings

This topic tells you how to install Service Bindings from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Service Bindings. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Service Bindings:

- Complete all prerequisites to install Tanzu Application Platform (commonly knows as TAP). For more information, see Prerequisites.

Install Service Bindings

Use the following procedure to install Service Bindings:

1. List version information for the package by running:

```
tanzu package available list service-bindings.labs.vmware.com --namespace tap-i
nstall
```

For example:

```
$ tanzu package available list service-bindings.labs.vmware.com --namespace tap
-install
- Retrieving package versions for service-bindings.labs.vmware.com...
  NAME                              VERSION  RELEASED-AT
  service-bindings.labs.vmware.com  0.5.0    2021-09-15T00:00:00Z
```

2. Install the package by running:

```
tanzu package install service-bindings -p service-bindings.labs.vmware.com -v
0.5.0 -n tap-install
```

Example output:

```
/ Installing package 'service-bindings.labs.vmware.com'
| Getting namespace 'tap-install'
- Getting package metadata for 'service-bindings.labs.vmware.com'
| Creating service account 'service-bindings-tap-install-sa'
| Creating cluster admin role 'service-bindings-tap-install-cluster-role'
| Creating cluster role binding 'service-bindings-tap-install-cluster-rolebindi
ng'
\ Creating package resource
| Package install status: Reconciling

 Added installed package 'service-bindings' in namespace 'tap-install'
```

3. Verify the package install by running:

```
tanzu package installed get service-bindings -n tap-install
```

Example output:

```
- Retrieving installation details for service-bindings...
NAME:                    service-bindings
PACKAGE-NAME:            service-bindings.labs.vmware.com
PACKAGE-VERSION:         0.5.0
STATUS:                  Reconcile succeeded
```

```
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

4. Run the following command:

   ```
   kubectl get pods -n service-bindings
   ```

   For example:

   ```
   $ kubectl get pods -n service-bindings
   NAME                        READY   STATUS    RESTARTS   AGE
   manager-6d85fffbcd-j4gvs   1/1     Running   0          22s
   ```

   Verify that `STATUS` is `Running`

Troubleshoot Service Bindings

This topic tells you how to troubleshoot Service Bindings in Tanzu Application Platform (commonly known as TAP).

Collect logs

To help identify issues when troubleshooting, you can retrieve and examine logs from the service binding manager.

To retrieve pod logs from the `manager` running in the `service-bindings` namespace, run:

```
kubectl -n service-bindings logs -l role=manager
```

For example:

```
$ kubectl -n service-bindings logs -l role=manager

2021/11/05 15:25:28 Registering 3 clients
2021/11/05 15:25:28 Registering 3 informer factories
2021/11/05 15:25:28 Registering 7 informers
2021/11/05 15:25:28 Registering 8 controllers
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.483823208Z","caller":"logging/nfi
g.go:116","message":"Successfully created the logger."}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.48392361Z","caller":"logging/confi
g.go:117","message":"Logging level set to: info"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.483999911Z","caller":"logging/conf
ig.go:79","message":"Fetch GitHub commit ID from kodata failed","error":"open /var/ru
n/ko/HEAD: no such file or directory"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.484035711Z","logger":"webhook","ca
ller":"profiling/server.go:64","message":"Profiling enabled: false"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.5228849092","logger":"webhook","ca
ller":"leaderelection/context.go:46","message":"Running with Standard leader electio
n"}
{"severity":"INFO","timestamp":"2021-11-05T15:25:28.523358615Z","logger":"webhook","ca
ller":"provisionedservice/controller.go:31","message":"Setting up event handlers."}
...
{"severity":"ERROR","timestamp":"2021-11-17T12:30:24.557178813Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"276.504µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T12:47:04.558217679Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"249.103µ
```

```
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:03:44.558683121Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"177.403µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:20:24.559192644Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"223.203µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:37:04.559648412Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"173.003µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T13:53:44.56010516Z","logger":"webhook","ca
ller":"controller/controller.go:548","message":"Reconcile error","duration":"182.402µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T14:10:24.560536033Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"155.603µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T14:27:04.560960243Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"171.002µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T14:43:44.56142548Z","logger":"webhook","ca
ller":"controller/controller.go:548","message":"Reconcile error","duration":"179.203µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
{"severity":"ERROR","timestamp":"2021-11-17T15:00:24.561881861Z","logger":"webhook","c
aller":"controller/controller.go:548","message":"Reconcile error","duration":"167.902µ
s","error":"deployments.apps \"spring-petclinic\" not found","stacktrace":"knative.de
v/pkg/controller.(*Impl).handleErr\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90db
```

```
b0/controller/controller.go:548\nknative.dev/pkg/controller.(*Impl).processNextWorkIte
m\n\tknative.dev/pkg@v0.0.0-20210331065221-952fdd90dbb0/controller/controller.go:531\n
knative.dev/pkg/controller.(*Impl).RunContext.func3\n\tknative.dev/pkg@v0.0.0-20210331
065221-952fdd90dbb0/controller/controller.go:468"}
```

Service Bindings resource specification

This topic tells you about the Service Bindings resource specification in Tanzu Application Platform (commonly known as TAP).

The `ServiceBinding` resource shape and behavior is defined by the following specification:

```
apiVersion: servicebinding.io/v1alpha3
kind: ServiceBinding
metadata:
  name: account-db
spec:
  service:
    apiVersion: mysql.example/v1alpha1
    kind: MySQL
    name: account-db
  workload:
    apiVersion: apps/v1
    kind: Deployment
    name: account-service
```

Overview of Services Toolkit

Services Toolkit is responsible for backing many of the most powerful service capabilities in Tanzu Application Platform (commonly known as TAP).

From the integration of an extensive list of cloud-based and on-prem services, through to the offering and discovery of those services, and finally to the claiming and binding of service instances to application workloads, Services Toolkit has the tools you need to make working with services on Tanzu Application Platform simple, easy, and effective.

Note

These docs apply to Services Toolkit v0.10 and later. To view the Services Toolkit documentation for v0.9 and earlier, see the previous Services Toolkit site.

Capabilities

The main capabilities on offer in Tanzu Application Platform through Services Toolkit are:

1. The classes and claims abstraction: provides a simple, but powerful, user experience to apps teams, while promoting a strong separation of concerns between apps teams and ops teams.
2. Dynamic provisioning of service instances: enables apps teams to create service instances that adhere to company policy. Apps teams can create instances on-demand as needed.
3. Seamless integration of almost any service, cloud-based or on-prem, into Tanzu Application Platform with minimal configuration overhead: provides a near-limitless range of services to help boost developer productivity.

Getting started

If this is your first time working with services on Tanzu Application Platform, you might want to start with Claim services on Tanzu Application Platform and Consume services on Tanzu Application Platform in the getting started guide. These guides run through the basics, after which you can return here to the Services Toolkit component documentation to continue your services journey on Tanzu Application Platform.

How this documentation is organized

The Services Toolkit component documentation consists of the following sections that relate to what you are want to achieve:

- Concepts: To gain a deeper understanding of Services Toolkit.
- Tutorials: To learn through following examples.
- How-to guides: To find a set of steps to solve a specific problem.
- Reference material: To find specific information such as Services Toolkit's APIs, the Tanzu Service CLI plug-in, and troubleshooting information.

Tutorials and concepts are of most relevance when studying, while how-to guides and reference material are of most use while working.

The following is a selection of useful topics on offer:

For apps teams:

- Tutorial: Working with Bitnami Services

For ops teams:

- Tutorial: Setup Dynamic Provisioning of Service Instances
- Tutorial: Integrating Cloud Services (AWS, Azure, GCP, etc.) into Tanzu Application Platform
- How-to guide: Configure Dynamic Provisioning of AWS RDS Service Instances

For everyone:

- Concept: Four Levels of Service Consumption in Tanzu Application Platform

Overview of Services Toolkit

Services Toolkit is responsible for backing many of the most powerful service capabilities in Tanzu Application Platform (commonly known as TAP).

From the integration of an extensive list of cloud-based and on-prem services, through to the offering and discovery of those services, and finally to the claiming and binding of service instances to application workloads, Services Toolkit has the tools you need to make working with services on Tanzu Application Platform simple, easy, and effective.

Note

These docs apply to Services Toolkit v0.10 and later. To view the Services Toolkit documentation for v0.9 and earlier, see the previous Services Toolkit site.

Capabilities

The main capabilities on offer in Tanzu Application Platform through Services Toolkit are:

1. The classes and claims abstraction: provides a simple, but powerful, user experience to apps teams, while promoting a strong separation of concerns between apps teams and ops

teams.

2. Dynamic provisioning of service instances: enables apps teams to create service instances that adhere to company policy. Apps teams can create instances on-demand as needed.
3. Seamless integration of almost any service, cloud-based or on-prem, into Tanzu Application Platform with minimal configuration overhead: provides a near-limitless range of services to help boost developer productivity.

Getting started

If this is your first time working with services on Tanzu Application Platform, you might want to start with Claim services on Tanzu Application Platform and Consume services on Tanzu Application Platform in the getting started guide. These guides run through the basics, after which you can return here to the Services Toolkit component documentation to continue your services journey on Tanzu Application Platform.

How this documentation is organized

The Services Toolkit component documentation consists of the following sections that relate to what you are want to achieve:

- Concepts: To gain a deeper understanding of Services Toolkit.
- Tutorials: To learn through following examples.
- How-to guides: To find a set of steps to solve a specific problem.
- Reference material: To find specific information such as Services Toolkit's APIs, the Tanzu Service CLI plug-in, and troubleshooting information.

Tutorials and concepts are of most relevance when studying, while how-to guides and reference material are of most use while working.

The following is a selection of useful topics on offer:

For apps teams:

- Tutorial: Working with Bitnami Services

For ops teams:

- Tutorial: Setup Dynamic Provisioning of Service Instances
- Tutorial: Integrating Cloud Services (AWS, Azure, GCP, etc.) into Tanzu Application Platform
- How-to guide: Configure Dynamic Provisioning of AWS RDS Service Instances

For everyone:

- Concept: Four Levels of Service Consumption in Tanzu Application Platform

Install Services Toolkit

This topic tells you how to install Services Toolkit from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Services Toolkit. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Services Toolkit:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager. For more information, see Install cert-manager.

Install Services Toolkit

To install Services Toolkit:

1. See what versions of Services Toolkit are available to install by running:

   ```
   tanzu package available list -n tap-install services-toolkit.tanzu.vmware.com
   ```

   For example:

   ```
   $ tanzu package available list -n tap-install services-toolkit.tanzu.vmware.com
   - Retrieving package versions for services-toolkit.tanzu.vmware.com...
     NAME                              VERSION           RELEASED-AT
     services-toolkit.tanzu.vmware.com  0.9.0             2022-09-08T00:00:00Z
   ```

2. Install Services Toolkit by running:

   ```
   tanzu package install services-toolkit \
     --package services-toolkit.tanzu.vmware.com \
     --version VERSION-NUMBER \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the Services Toolkit version you want to install. For example, `0.9.0`.

3. Verify that the package installed by running:

   ```
   tanzu package installed get services-toolkit -n tap-install
   ```

   In the output, confirm that the `STATUS` value is `Reconcile succeeded`.

   For example:

   ```
   $ tanzu package installed get services-toolkit -n tap-install
   | Retrieving installation details for services-toolkit...
   NAME:                    services-toolkit
   PACKAGE-NAME:            services-toolkit.tanzu.vmware.com
   PACKAGE-VERSION:         0.9.0
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

Services Toolkit concepts

This section introduces you to Services Toolkit concepts.

In this section:

- The four levels of service consumption in Tanzu Application Platform
- Class claims vs resource claims

The four levels of service consumption in Tanzu Application Platform

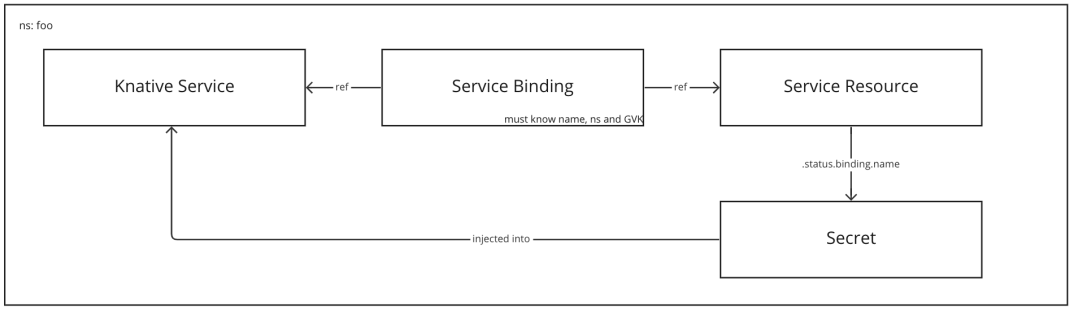
This topic describes the different levels of abstraction when using Services Toolkit and explains when and why you might choose to use one level over the other.

As Tanzu Application Platform has evolved, so has the way to offer and consume services on the platform. In this topic, the progress of this evolution is charted in terms of four levels of service consumption.

The introduction of a higher level does not automatically mean that all lower levels are made obsolete. In most cases, the higher levels build upon the foundations laid by the lower levels, and represent an abstraction that is higher-level and more opinionated.

Level 1 - direct bindings

Tanzu Application Platform v1.0 included support for service bindings, which back the level 1 concept of direct bindings.



For example, if you deploy an application workload to Tanzu Application Platform, this results in a Knative service in a namespace. An API resource in the same namespace that represents a service, such as a database or a cache, is called a service resource.

Using a service binding you can bind that service resource with the Knative service. This injects the credentials for the service resource into the Knative service, so that the application workload can consume it.

In this relatively straightforward scenario, there are only a few resources involved and they directly reference each other. In fact, users are not even directly exposed to the service binding. The service binding is created automatically as part of the Out of the Box Supply Chains whenever an application workload is configured to refer to a service.

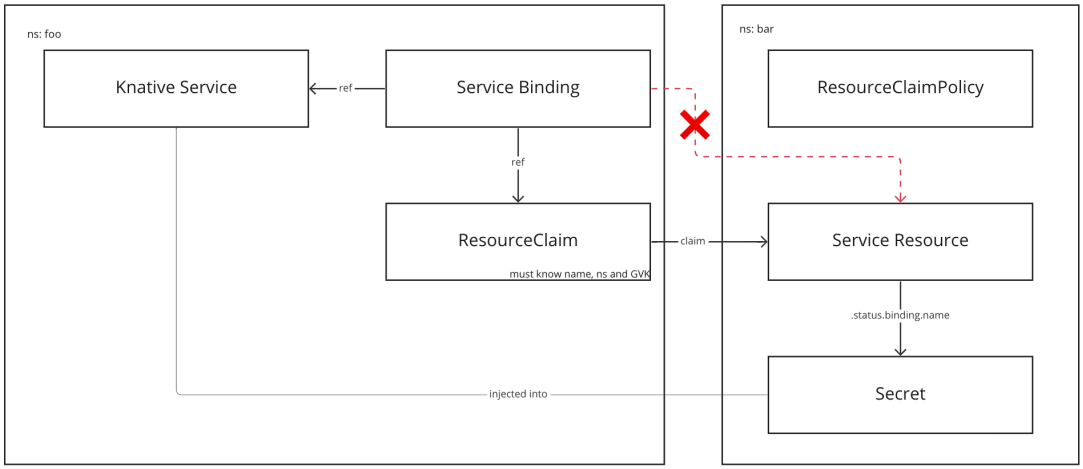
However, there are a number of limitations with this setup. The first is that the service resource must be bindable and which means it must adhere to the provisioned services definition in the service binding specification for Kubernetes. For more information, see Provisioned Service. There are some resources that adhere to this specification, primarily resources offered by VMware Tanzu's data services, but the overwhelming majority of resources don't.

The second limitation is that all resources have to be in the same namespace.

The third limitation is that the service binding must have detailed and specific information about the service resource, including its name, namespace, and API group, version, and kind. This is not a clear separation of concerns as it introduces tight coupling between app teams, who create the application workloads, and ops teams, who create the service resources.

Level 2 - resource claims

Level 2 addresses some limitations of direct bindings through the Services Toolkit feature resource claims. This feature coincided with the release of Tanzu Application Platform v1.0.



For example, if you have the same setup as in Level 1 - direct bindings, but you want the service resource to be in a separate namespace from the Knative service you cannot use direct bindings. This is because the schema for ServiceBinding provides no option to configure a `namespace` for the service.

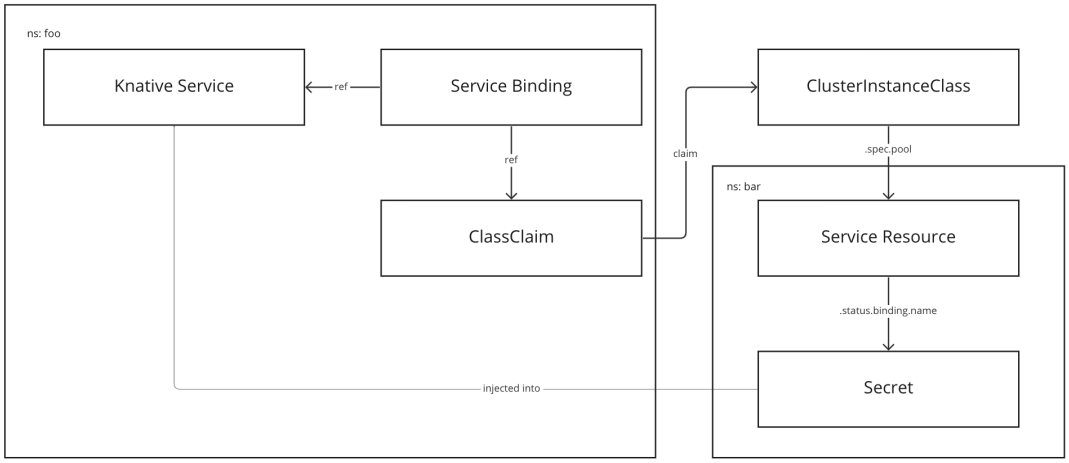
Resource claims allow you to claim a bindable service resource that exists in another namespace, and to then bind the application workload to the resource claim instead of to the service resource directly. Because you are now crossing namespace tenancy boundaries, you are only permitted to claim the service resource if you create a corresponding resource claim policy.

The advantage of level 2 over level 1 is that now the application workload and the service resource do not have to exist in the same namespace. This helps to promote a better separation of concerns. It is now possible for apps teams and ops teams to manage the life cycles of apps and services independently.

However, it's still not an ideal solution, and some limitations from level 1 still exist in level 2. The service resource still must be bindable, and apps teams still must know the name, namespace, and API group, version, and kind of the service resource. In addition, ops teams must ensure that the service resources exist. These resources must be manually provisioned and permitted to be claimed through policy, otherwise resource claims created by the apps teams remain in a pending state indefinitely.

Level 3 - class claims and pool-based classes

Level 3 introduces class claims and pool-based classes. Originally released with Tanzu Application Platform v1.4, class claims and pool-based classes help to alleviate the issue of apps teams having to know detailed information about service resources.



Level 3 builds on the example in level 2, which has an application workload in a namespace `foo` and a service resource in a namespace `bar`. Rather than relying on resource claim and resource claim policy, level 3 introduces a class claim and a pool-based class.

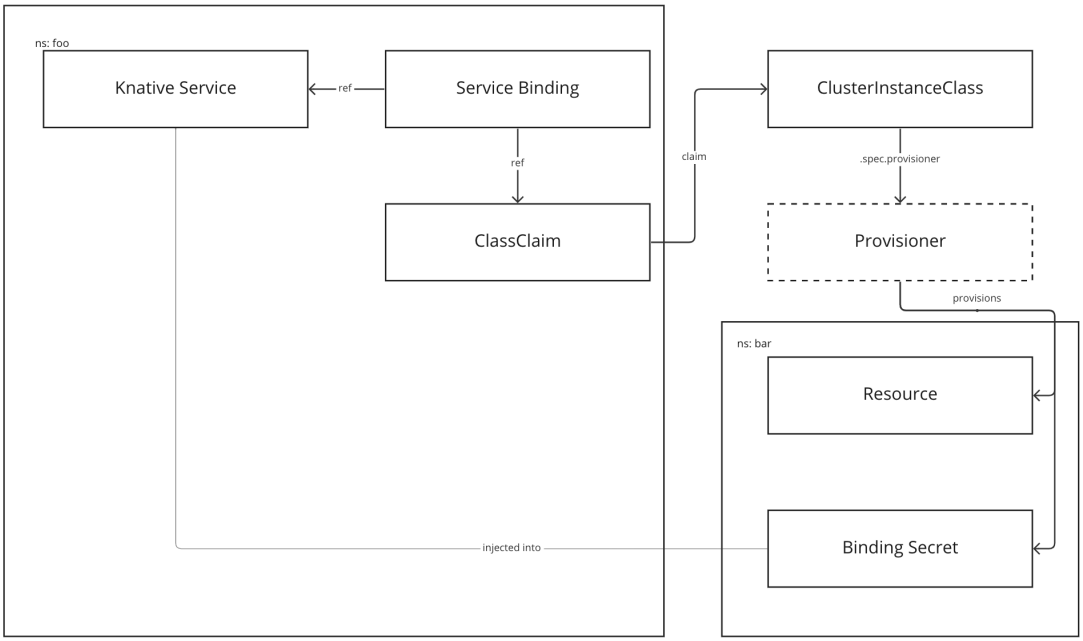
The pool-based class can pool service resources using label or field selectors from across all namespaces on the cluster. Ops teams create the service resources and then create a class to gather them all together into one logical group that apps teams can discover and claim from.

Apps teams can discover the available classes using the `tanzu service class list` command. Rather than creating a resource claim, they instead create a higher-level abstraction - a class claim. The class claim refers to the name of a class. You only need to create a class claim referring to a class and then bind your application workload to the class claim. There is no longer a need to provide detailed information such as the API group, version, and kind for the service resource behind the class.

Level 3 is much simpler for apps teams to consume services and the separation of concerns is much neater. A few limitations still remain. Service resources must still be bindable and ops teams still must manually provision the service resources to fill the pool.

Level 4 - class claims and provisioner-based classes (aka “Dynamic Provisioning”)

Level 4 is the current highest level of service consumption in Tanzu Application Platform. Released in Tanzu Application Platform v1.5, it introduces provisioner-based classes, which, together with class claims, power Tanzu Application Platform’s dynamic provisioning capability.



Level 4 builds on the example in level 3, but the class now defines a provisioner rather than a pool. Services Toolkit in Tanzu Application Platform v1.5 supports one provisioner type - Crossplane. Support for new provisioners might be added in the future.

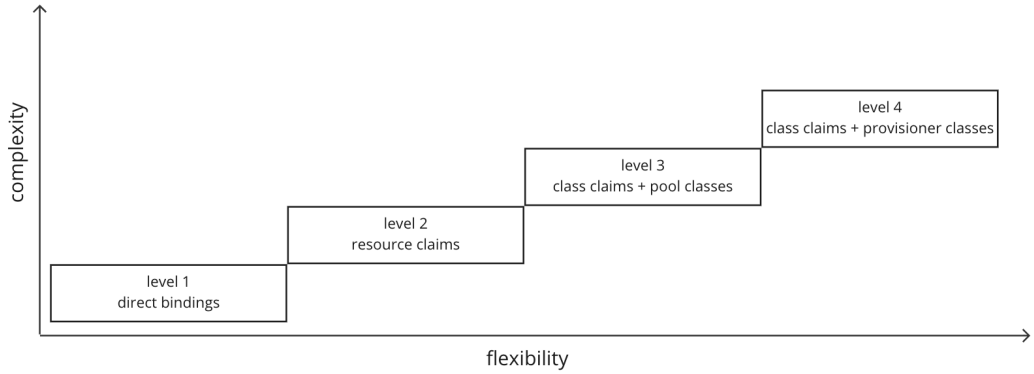
When you create a class claim that refers to a provisioner-based class, the Services Toolkit controller requests the provisioner to provision the resources necessary to create a service instance that can then be bound to application workloads. In this example, the provisioner creates a namespace, a service resource, and a secret that conforms to the binding specification. Then, that secret is wired all the way back through to the application workload.

Two big advantages are realized at level 4. Firstly, ops teams no longer need to manually provision service resources. They are now created on-demand as and when needed. This not only helps to remove unnecessary burden from ops teams, but also helps to provide better use of resources because service resources no longer need to remain in a pool waiting to be claimed. The second advantage is that service resources no longer need to be bindable. The provisioner can act almost like an adapter to bring pretty much any service you can think of into Tanzu Application Platform. The only requirement is that the provisioner create a binding-conforming secret that holds credentials for the provisioned service resources. You can configure this once during the dynamic provisioning setup.

While level 4 is very powerful and seemingly solves the problems mentioned so far, it's not entirely without its drawbacks. The main one being that all this flexibility comes at the cost of added complexity. There are many, many more moving parts involved at level 4 when compared to level 1. However, it might be a price worth paying to benefit from all that is on offer.

Summary

By now you have seen how each new level builds and improves upon the last. All levels are also valid use cases in their own right.



Tanzu Application Platform users can decide which level to operate at. Finding the level that is right for your situation depends on a number of factors, such as, the size of the organization you work for and the layout of apps teams and ops teams within the org.

If you work at a small startup in which there are no strict divides between apps teams and ops teams, level 1 might be suitable for your needs. However, if you work for a large organization with distinct and dedicated apps and ops teams, choosing one of the higher levels in which that separation is better catered to might make more sense. If you are not sure, it's probably best to start with level 4. Level 4 provides the ultimate services experience on Tanzu Application Platform, and as such will hopefully meet all your services needs.

Class claims compared to resource claims

There are two types of claim you can choose from when working with services on Tanzu Application Platform (commonly known as TAP). These are ClassClaim and ResourceClaim. This Services Toolkit topic explains the similarities and differences between the two and when using one is preferable over the other.

It is usually advisable to work with a ClassClaim where possible as they are easier to create and are more portable across multiple clusters. They are also used as the trigger mechanism for dynamic provisioning of service instances.

Similarities

- Both APIs express that you want to access to a service instance.
- Both APIs adhere to the ProvisionedService duck type. They both have the field `.status.binding.name` in their API. This means that you can target them using a ServiceBinding and, therefore, you can feed them into Cartographer's Workload API.
- Both APIs ensure that mutual exclusivity of claims on service instances. After using either a ClassClaim or a ResourceClaim to claim a service instance, no other ClassClaim or a ResourceClaim can claim that same service instance.

Using a ResourceClaim

A ResourceClaim targets a specific resource in the Kubernetes cluster. To target that resource, the ResourceClaim needs the name, namespace, kind, and API version of the resource.

The specificity of the ResourceClaim means it is most useful when you must guarantee which service instance the application workload uses. For example, if the application must connect to the exact same database instance while it advances through development, test, and production

environments. If you do not need this guarantee VMware recommends that you use the ClassClaim API instead.

Using a ClassClaim

A ClassClaim targets a ClusterInstanceClass in the Kubernetes cluster. To target this class, the ClassClaim only requires the name of the ClusterInstanceClass.

The ClusterInstanceClass can represent any set of service instances and therefore each time you create a new ClassClaim, you can claim any of the service instances represented by that ClusterInstanceClass. After a ClassClaim has claimed a service instance, it never looks for another. This is true even if the ClassClaim's `spec` is updated, or the ClusterInstanceClass is updated. Therefore, the ClassClaim is performing a **point-in-time** lookup at its creation, using the ClusterInstanceClass for that lookup.

The loose coupling between the ClassClaim and the service instances means that a ClassClaim is best in situations where:

- You must inject different service instances into the application workload at different points in its advancement from development to production environments. For more information, see Abstracting Service Implementations Behind A Class Across Clusters.
- The ClassClaim, and also any workload referencing it, must be promoted from one environment to the next without changing their specification.

The ClassClaim is the only type of claim that you can use to dynamically provision service instances.

Tutorials

In this section:

For apps teams:

- Working with Bitnami Services

For ops teams:

- Set up dynamic provisioning of service instances
- Integrating cloud services into Tanzu Application Platform
- Using direct secret references
- Abstracting service implementations behind a class across clusters

Set up dynamic provisioning of service instances

In this Services Toolkit tutorial you learn how service operators can set up a new, self-serve, and customized service for Tanzu Application Platform (commonly known as TAP). The example uses VMware RabbitMQ for Kubernetes, but the steps and learnings can apply to almost any other service.

About this tutorial

Target user role: Service Operator
Complexity: Advanced
Estimated time: 60 minutes
Topics covered: Dynamic Provisioning, Crossplane, VMware RabbitMQ for Kubernetes operator
Learning outcomes: Ability to offer new, on-demand, and customized services in your Tanzu Application Platform clusters

Prerequisites

- Access to a Tanzu Application Platform cluster v1.5.0 or later.
- Basic familiarity with Crossplane, particularly the concepts of Composition and CompositeResourceDefinitions.

Scenario

The tutorial is centered around the following hypothetical, but somewhat realistic, real-world scenario.

You work at BigCorp and are tasked to provide an on-demand, self-serve RabbitMQ service for BigCorp's development teams who are working with Tanzu Application Platform. You have already reviewed the RabbitMQ offering that is available with Bitnami Services, but have discovered that while it is an excellent service for testing and for quickly getting started, it is not quite suitable for BigCorp's stringent and specific needs.

In particular, you must comply with BigCorp's auditing and logging policy, and want to enforce that every RabbitMQ cluster in use on the platform adheres to that policy. At the same time, you don't want to be a blocker for the application teams and want to offer them self-serve access to RabbitMQ whenever they need it, without incurring any untoward delays. You have heard great things about Tanzu Application Platform's dynamic provisioning capability, and are now looking to make use of it to help you complete your task.

In this tutorial you will learn how to:

- Install the RabbitMQ Cluster Kubernetes operator
- Create a `CompositeResourceDefinition`
- Create a `Composition`
- Create a provisioner-based class
- Understand and create the necessary RBAC permissions
- Create a claim for the class to test it all out
- Understand how all the pieces fit together to power the dynamic provisioning capability in Tanzu Application Platform

Concepts

The following diagram provides an overview of the elements of dynamic provisioning and how they fit together.

The following is a high-level overview of how the system works:

1. The service operator creates a `CompositeResourceDefinition` and a `Composition`, which together define the configuration of the service instances that will be dynamically provisioned.
2. The service operator creates a class pointing to the `CompositeResourceDefinition`. This informs application development teams that the service is available.
3. The service operator applies necessary Role-Based Access Control (RBAC) to permit the system to create the necessary resources, and to authorize application development teams to create claims for the class.
4. The application developer creates a claim referring to the class, optionally passing through parameters to override any default configuration where permissible.
5. The system creates a `CompositeResource`, merging information provided in the claim with default configuration specified by the system and configuration defined in the `Composition`.
6. Crossplane reconciles the `CompositeResource` into a service instance and writes credentials for the instance into a `Secret`.
7. The `Secret` is written back to the application developer's namespace, so that application workloads can use it.

As you follow this tutorial, it will address the parts of this diagram in more detail.

Procedure

The following steps show how to configure dynamic provisioning for a service.

Step 1: Install the operator

When adding any new service to Tanzu Application Platform, ensure that there are a suitable set of APIs available in the cluster from which to construct the service instances. Usually, this involves installing one or more Kubernetes Operators into the cluster.

Given the aim of this tutorial is to set up a new RabbitMQ service, install the RabbitMQ Cluster Operator for Kubernetes.

Note

> The steps in this tutorial use the open source version of the operator. For most real-world deployments, VMware recommends using the official, supported version provided by VMware. For more information, see VMware RabbitMQ for Kubernetes.

Use `kapp` to install the operator by running:

```
kapp -y deploy --app rmq-operator --file https://github.com/rabbitmq/cluster-operator/releases/latest/download/cluster-operator.yml
```

This causes a new API Group/Version of `rabbitmq.com/v1beta1` and Kind named `RabbitmqCluster` to become available in the cluster. You can now use this API to create RabbitMQ cluster instances as part of the dynamic provisioning setup.

Step 2: Creating a `CompositeResourceDefinition`

Tanzu Application Platform's dynamic provisioning capability relies on Crossplane. You can find the specific integration point at `.spec.provisioner.crossplane.compositeResourceDefinition` in Tanzu Application Platform's `ClusterInstanceClass` API.

As the name suggests, this field is looking for a `CompositeResourceDefinition`, which you create in this step of the procedure. The `CompositeResourceDefinition` (XRD) defines the shape of a new, custom API type that encompasses the specific set of requirements laid out by the scenario in this tutorial.

Create a file named `xrabbitmqclusters.messaging.bigcorp.org.xrd.yaml` and copy in the following contents.

```
# xrabbitmqclusters.messaging.bigcorp.org.xrd.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: CompositeResourceDefinition
metadata:
  name: xrabbitmqclusters.messaging.bigcorp.org
spec:
  connectionSecretKeys:
  - host
  - password
  - port
  - provider
  - type
  - username
  group: messaging.bigcorp.org
  names:
    kind: XRabbitmqCluster
    plural: xrabbitmqclusters
  versions:
  - name: v1alpha1
    referenceable: true
    schema:
      openAPIV3Schema:
        properties:
          spec:
            description: The OpenAPIV3Schema of this Composite Resource Definition.
            properties:
              replicas:
                description: The desired number of replicas forming the cluster
                type: integer
              storageGB:
                description: The desired storage capacity of a single replica, in GB.
```

```
                type: integer
            type: object
        type: object
    served: true
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f xrabbitmqclusters.messaging.bigcorp.org.xrd.yaml
```

For a detailed explanation of `CompositeResourceDefinition` see, the Crossplane documentation.

The following is a condensed explanation of the most relevant pieces of the `CompositeResourceDefinition` configuration, provided in this section, as it relates to dynamic provisioning in Tanzu Application Platform.

The example in this tutorial does **not** specify `.spec.claimNames` in the XRD. Tanzu Application Platform's dynamic provisioning capability makes use of Crossplane's cluster-scoped Composite Resources, rather than the namespace-scoped Claims ("Claims" here not to be confused with Tanzu Application Platform's own concept of claims). As such, this configuration is not required, although it does not cause any adverse effects if you add it.

Next, see the `.spec.connectionKeys` field. This field detects the keys that will exist in the `Secret` resulting from the dynamic provisioning request. You likely want this `Secret` to conform with the Service Binding Specification for Kubernetes, as this, in part, is what allows for automatic configuration of the service instance by Tanzu Application Platform's application workloads. This is assuming that the application is using a binding-aware library such as Spring Cloud Bindings. Specific key name requirements vary by service type, however all must provide the `type` key.

Finally, see the `.spec.properties` section in the schema for `v1alpha1`. This is where you, as the service operator, can set which configuration options you want to expose to application development teams. In the example in this section, there are two configuration options: `replicas` and `storageGB`. By adding these properties to the specification, you are handing over control of these specific configuration options to the development teams. For example, you might want to add `storageGB` if the development teams have more knowledge about how much storage their apps require than you do. By adding `storageGB` you can allow them to decide for themselves how much storage they require.

You can choose to add as many or as few configuration options here as you like. You can also choose to set default values. In highly regulated environments, you might not want to allow for any configuration by developers at all.

In the scenario at the beginning of this tutorial, it says that you must comply with the auditing and logging policy. You do not specify any configuration related to auditing or logging in the XRD in this step. This is intentional as in this scenario there are strict auditing and logging requirements and cannot permit developers to override those. In the next step you learn how to ensure that those requirements get enforced on the resulting RabbitMQ clusters.

To verify the status of the XRD you created, run:

```
kubectl get xrds
```

If successful, the `xrabbitmqclusters.messaging.bigcorp.org` is listed with `ESTABLISHED=True`.

You might see some other XRDs listed as well. These are the `*.bitnami.*.tanzu.vmware.com` XRDs. These are part of the `bitnami.services.tanzu.vmware.com` package with Tanzu Application Platform and serve as the basis of the Bitnami Services. You can ignore these other XRDs for now, but if you want to see how they are used in practice, see Claim services on Tanzu Application Platform and Consume services on Tanzu Application Platform in the Tanzu Application Platform getting started guide.

As a result of creating the XRD, a new API Group/Version of `messaging.bigcorp.org/v1alpha1` and Kind named `XRabbitmqCluster` become available in the cluster. If you inspect this API further, notice that the `replicas` and `storageGB` properties configured in the XRD are present in the specification of `XRabbitmqCluster`.

```
kubectl explain --api-version=messaging.bigcorp.org/v1alpha1 xrabbitmqclusters.spec
```

You will also notice that Crossplane has injected some other fields into the specification as well, but you can mostly ignore these for now.

Step 3: Creating a Crossplane `Composition`

You do most of the configuration for dynamic provisioning during the creation of the `Composition`.

For a more detailed explanation about the `Composition`, see the Crossplane documentation.

The following are the basics you must know to start to create a `Composition` for use in Tanzu Application Platform.

Create a file named `xrabbitmqclusters.messaging.bigcorp.org.composition.yaml` and copy in the following contents.

```
# xrabbitmqclusters.messaging.bigcorp.org.composition.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: xrabbitmqclusters.messaging.bigcorp.org
spec:
  compositeTypeRef:
    apiVersion: messaging.bigcorp.org/v1alpha1
    kind: XRabbitmqCluster
  resources:
  - base:
      apiVersion: kubernetes.crossplane.io/v1alpha1
      kind: Object
      spec:
        forProvider:
          manifest:
            apiVersion: rabbitmq.com/v1beta1
            kind: RabbitmqCluster
            metadata:
              namespace: rmq-clusters
            spec:
              terminationGracePeriodSeconds: 0
              replicas: 1
              persistence:
                storage: 1Gi
              resources:
                requests:
                  cpu: 200m
                  memory: 1Gi
                limits:
                  cpu: 300m
                  memory: 1Gi
              rabbitmq:
                envConfig: |
                  RABBITMQ_LOGS=""
                additionalConfig: |
                  log.console = true
                  log.console.level = debug
                  log.console.formatter = json
                  log.console.formatter.json.field_map = verbosity:v time msg domain f
```

```
ile line pid level:-
                  log.console.formatter.json.verbosity_map = debug:7 info:6 notice:5 w
arning:4 error:3 critical:2 alert:1 emergency:0
                  log.console.formatter.time_format = epoch_usecs
        connectionDetails:
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.provider
          toConnectionSecretKey: provider
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.type
          toConnectionSecretKey: type
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.host
          toConnectionSecretKey: host
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.port
          toConnectionSecretKey: port
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.username
          toConnectionSecretKey: username
        - apiVersion: v1
          kind: Secret
          namespace: rmq-clusters
          fieldPath: data.password
          toConnectionSecretKey: password
        writeConnectionSecretToRef:
          namespace: rmq-clusters
    connectionDetails:
    - fromConnectionSecretKey: provider
    - fromConnectionSecretKey: type
    - fromConnectionSecretKey: host
    - fromConnectionSecretKey: port
    - fromConnectionSecretKey: username
    - fromConnectionSecretKey: password
    patches:
      - fromFieldPath: metadata.name
        toFieldPath: spec.forProvider.manifest.metadata.name
        type: FromCompositeFieldPath
      - fromFieldPath: spec.replicas
        toFieldPath: spec.forProvider.manifest.spec.replicas
        type: FromCompositeFieldPath
      - fromFieldPath: spec.storageGB
        toFieldPath: spec.forProvider.manifest.spec.persistence.storage
        transforms:
        - string:
            fmt: '%dGi'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.writeConnectionSecretToRef.name
        transforms:
        - string:
            fmt: '%s-rmq'
            type: Format
          type: string
```

```
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[0].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[1].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[2].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[3].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[4].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[5].name
        transforms:
        - string:
            fmt: '%s-default-user'
            type: Format
          type: string
        type: FromCompositeFieldPath
    readinessChecks:
      - type: MatchString
        fieldPath: status.atProvider.manifest.status.conditions[1].status # ClusterAva
ilable
        matchString: "True"
```

Use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f xrabbitmqclusters.messaging.bigcorp.org.composition.yaml
```

About `.spec.compositeTypeRef`

The `.spec.compositeTypeRef` is configured to refer to `XRabbitmqCluster` on the `messaging.bigcorp.org/v1alpha1` API group and version.

```
...
spec:
  compositeTypeRef:
    apiVersion: messaging.bigcorp.org/v1alpha1
    kind: XRabbitmqCluster
...
```

This is the API that was created when you applied the XRD in Step 2: Creating a `CompositeResourceDefinition`. By configuring `.spec.compositeTypeRef` to refer to this API, you are instructing Crossplane to use the configuration contained within this `Composition` to compose subsequent managed resources whenever it observes that a new `XRabbitmqCluster` resource is created in the cluster. Tanzu Application Platform's dynamic provisioning system creates the `XRabbitmqCluster` resources automatically. To visualize how these pieces fit together, see the diagram in the Concepts section.

About `.spec.resources`

The `.spec.resources` section is where you specify the managed resources to be created. Managed resources are tied to Crossplane's `Providers`, with each `Provider` defining a set of managed resources which can then be used in compositions. Tanzu Application Platform includes two `Providers` with the Crossplane package: provider-helm and provider-kubernetes. This makes a `Release` managed resource available, which is used to manage Helm releases, and makes an `Object` managed resource available, which used to manage arbitrary Kubernetes resources. You can install and use any other `Provider`. To find the latest providers, see the Upbound Marketplace. The more providers you install, the more managed resources you can choose from in your compositions.

The `Object` managed resource

The overarching goal is to compose whatever resources are necessary to create functioning, usable service instances and to surface the credentials and connectivity information required to connect to those instances in a known and repeatable way. This tutorial uses the `RabbitmqCluster` resource, which presents one single API to use to create fully functioning RabbitMQ clusters, credentials for which get stored in `Secrets` in the cluster.

However, `RabbitmqCluster` is not a Crossplane managed resource so you cannot refer to this resource directly under `.spec.resources`. To work around this, use `provider-kubernetes` and its corresponding `Object` managed resource. `Object` enables you to wrap any arbitrary Kubernetes resource, such as `RabbitmqCluster`, into a Crossplane managed resource and to then use them like any other managed resource inside `Compositions`.

```
...
spec:
  resources:
  - base:
      apiVersion: kubernetes.crossplane.io/v1alpha1
      kind: Object
      spec:
        forProvider:
          manifest:
            apiVersion: rabbitmq.com/v1beta1
            kind: RabbitmqCluster
            metadata:
              namespace: rmq-clusters
            spec:
              terminationGracePeriodSeconds: 0
              replicas: 1
              persistence:
                storage: 1Gi
```

```
              resources:
                requests:
                  cpu: 200m
                  memory: 1Gi
                limits:
                  cpu: 300m
                  memory: 1Gi
              rabbitmq:
                envConfig: |
                  RABBITMQ_LOGS=""
                additionalConfig: |
                  log.console = true
                  log.console.level = debug
                  log.console.formatter = json
                  log.console.formatter.json.field_map = verbosity:v time msg domain f
ile line pid level:-
                  log.console.formatter.json.verbosity_map = debug:7 info:6 notice:5 w
arning:4 error:3 critical:2 alert:1 emergency:0
                  log.console.formatter.time_format = epoch_usecs
...
```

The `Object` managed resource is where you configure `RabbitmqCluster` resources. This is the place in which you can now fine-tune the configuration of the RabbitMQ Clusters to your needs.

Recall from the hypothetical scenario that you are particularly concerned about your company's logging policy. The configuration in the `Object` translates that hypothetical policy into default configuration on the `RabbitmqCluster` resource by specifying `.spec.rabbitmq.additionalConfig` for the resource. This was taken from one of the examples in the RabbitMQ Cluster Operator GitHub repository. You can configure the resource however you want and to whatever requirements necessary.

The `patches` section

The `Object` also sets default values for the number of replicas and the amount of persistent storage for new `RabbitmqClusters` to one replica and 1 Gi. However, you want to allow these two values to be configurable by the application development teams as specified in Step 2: Creating a `CompositeResourceDefinition`. You can configure this using patches.

```
...
patches:
  - fromFieldPath: metadata.name
    toFieldPath: spec.forProvider.manifest.metadata.name
    type: FromCompositeFieldPath
...
```

The first thing to note is that all the patches are of type `FromCompositeFieldPath`, which allows you to take values defined on the composite resource (`XRabbitmqCluster` in this case) and to pass them through to the underlying managed resource (an `Object` wrapping `RabbitmqCluster` in this case). The first patch sets the name of the `RabbitmqCluster` to the same name as the name of the composite resource `XRabbitmqCluster`, which were created using `generateName`, thereby ensuring a unique name for each dynamically provisioned `RabbitmqCluster` instance.

```
...
patches:
...
  - fromFieldPath: spec.replicas
    toFieldPath: spec.forProvider.manifest.spec.replicas
    type: FromCompositeFieldPath
  - fromFieldPath: spec.storageGB
    toFieldPath: spec.forProvider.manifest.spec.persistence.storage
    transforms:
```

```
    - string:
        fmt: '%dGi'
        type: Format
      type: string
    type: FromCompositeFieldPath
...
```

The second and third patches pass through configuration for the number of replicas and amount of persistent storage, which overrides the default values already configured.

The remaining patches all do the same thing, which is to patch in the name of the `Secret` for the fields in the `connectionDetails` section.

```
...
- fromFieldPath: metadata.name
  toFieldPath: spec.connectionDetails[0].name
  transforms:
  - string:
      fmt: '%s-default-user'
      type: Format
    type: string
  type: FromCompositeFieldPath
...
```

When creating a `RabbitmqCluster` resource using the RabbitMQ Cluster Kubernetes operator, the operator creates a `Secret` containing credentials and connectivity information used to connect to the cluster. That `Secret` is named `x-default-user`, where `x` is the name of the `RabbitmqCluster` resource. Because the name of the `RabbitmqCluster` cannot be known upfront, you must use patches to ensure that the `connectionDetails` section refers to the correctly-named `Secret`.

The `connectionDetails` sections are where you configure which keys and values to expose in the resulting `Secret`. You must specify the same set of keys as defined in the original XRD.

The `readinessChecks` section

Configuring readiness checks helps to keep consumers of dynamic provisioning, that is, the application teams, informed about when the resulting service instances are ready for application workloads to use.

```
...
readinessChecks:
  - type: MatchString
    fieldPath: status.atProvider.manifest.status.conditions[1].status # ClusterAvailab
le
    matchString: "True"
```

Where possible it is simplest to use the `Ready` condition to verify readiness. However, the `RabbitmqCluster` API doesn't expose a simple `Ready` condition, so you must configure the ready check on `ClusterAvailable` instead.

Check the namespace

One final important decision is the name of the namespace in which to create the dynamically provisioned `RabbitmqCluster` resources. This tutorial uses the `rmq-clusters` namespace.

```
...
spec:
  resources:
  - base:
      apiVersion: kubernetes.crossplane.io/v1alpha1
```

```
      kind: Object
      spec:
        forProvider:
          manifest:
            apiVersion: rabbitmq.com/v1beta1
            kind: RabbitmqCluster
            metadata:
              namespace: rmq-clusters
...
```

To make sure that the `rmq-clusters` namespace exists.

```
kubectl create namespace rmq-clusters
```

This configuration says that all dynamically provisioned `RabbitmqCluster` resources must be placed in the same `rmq-clusters` namespace. If you want to place each new cluster into a separate namespace, you must create an additional `Object` managed resource to wrap the creation of a `Namespace` and to apply patches to the resources accordingly. For this tutorial you only require one namespace.

Step 4: Creating a provisioner-based class

The creation of the XRD and the Composition brings to an end the Crossplane-centric part of this tutorial. What remains is to integrate all that you configured into Tanzu Application Platform's classes and claims model so that application teams can more easily make use of it. The first step here is to create a provisioner-based class and to point it at the XRD you created.

Create a file named `bigcorp-rabbitmq.class.yaml` and copy in the following contents.

```
# bigcorp-rabbitmq.class.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: bigcorp-rabbitmq
spec:
  description:
    short: On-demand RabbitMQ clusters precision engineered to meet the needs of BigCo
rp!
  provisioner:
    crossplane:
      compositeResourceDefinition: xrabbitmqclusters.messaging.bigcorp.org
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f bigcorp-rabbitmq.class.yaml
```

This is referred to as a provisioner-based class due to the configuration of `.spec.provisioner`. For more information, see ClusterInstanceClass.

By creating this class you are informing application teams that the service is available. Application teams can discover it by using the `tanzu service class list` command. They can also use `tanzu service class get bigcorp-rabbitmq`, which provides detailed information about the class, including details of the `replicas` and `storageGB` parameters that you configured earlier.

Step 5: Configure supporting RBAC

There are two parts of RBAC to consider when you set up a new service for dynamic provisioning in Tanzu Application Platform. The first relates to granting permissions to the providers used in the compositions. The `Composition` created earlier uses `Object` managed resources ultimately to create

`RabbitmqCluster` resources. Therefore, you must grant `provider-kubernetes` permission to create `RabbitmqCluster` resources. You can do this by using an aggregating `ClusterRole` as follows.

Create a file named `provider-kubernetes-rmqcluster-read-writer.rbac.yaml` and copy in the following contents.

```
# provider-kubernetes-rmqcluster-read-writer.rbac.yaml

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rmqcluster-read-writer
  labels:
    services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"
rules:
- apiGroups:
  - rabbitmq.com
  resources:
  - rabbitmqclusters
  verbs:
  - "*"
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f provider-kubernetes-rmqcluster-read-writer.rbac.yaml
```

While not necessary here, a corresponding label `services.tanzu.vmware.com/aggregate-to-provider-helm: "true"` exists for aggregating RBAC permissions to `provider-helm` as well.

The second element of RBAC detects who is authorized to use the new service. This is an important piece of configuration. You are configuring an on-demand service and making it available to application teams. Without any other supporting policy in place, application teams can create as many `RabbitmqClusters` as they like. This is of course the whole point of an on-demand service, but you must be conscious of resource use, and might want to control who can create new service instances on-demand.

You can grant authorization by using standard Kubernetes RBAC resources. Dynamic provisioning uses a custom RBAC verb, `claim`, which you can apply to classes to permit claiming from classes.

Create a file named `app-operator-claim-class-bigcorp-rabbitmq.rbac.yaml` and copy in the following contents.

```
# app-operator-claim-class-bigcorp-rabbitmq.rbac.yaml

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-class-bigcorp-rabbitmq
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups:
  - services.apps.tanzu.vmware.com
  resources:
  - clusterinstanceclasses
  resourceNames:
  - bigcorp-rabbitmq
  verbs:
  - claim
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f app-operator-claim-class-bigcorp-rabbitmq.rbac.yaml
```

This `ClusterRole` grants anyone holding the `app-operator` Tanzu Application Platform user role the ability to claim from the `bigcorp-rabbitmq` class.

Step 6: Verify your configuration

To test your configuration, create a claim for the class and thereby trigger the dynamic provisioning of a new RabbitMQ cluster. This step is typically performed by the application operator, rather than the service operator, but it is important that you to confirm that everything is configured correctly.

Create a file named `bigcorp-rmq-1.claim.yaml` and copy in the following contents.

```
# bigcorp-rmq-1.claim.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: bigcorp-rmq-1
spec:
  classRef:
    name: bigcorp-rabbitmq
  parameters:
    storageGB: 2
    replicas: 3
```

Then use kubectl to apply the file to the Tanzu Application Platform cluster.

```
kubectl apply -f bigcorp-rmq-1.claim.yaml
```

After the RabbitMQ service is provisioned, the claim status reports `Ready=True`.

```
kubectl get classclaim bigcorp-rmq-1
```

Working with Bitnami Services

For the tutorial about working with Bitnami Services, see Working with Bitnami Services.

Integrating cloud services into Tanzu Application Platform

In this Services Toolkit tutorial you learn how service operators can integrate the cloud services of their choice into Tanzu Application Platform (commonly known as TAP).

There are a multitude of cloud-based services available on the market for consumers today. AWS, Azure, and GCP all provide support for a wide range of fully-managed, performant and on-demand services ranging from databases, to message queues, to storage solutions and beyond. In this tutorial you will learn how to integrate any one of these services into Tanzu Application Platform, so that you can offer it for apps teams to consume in a simple and effective way.

This tutorial is written at a slightly higher level than the other tutorials in this documentation. This is because it is not feasible to write detailed, step-by-step documentation for integrating every cloud-based service into Tanzu Application Platform. Each service brings a different set of considerations and concerns.

Instead, this tutorial guides you through the general approach to integrating cloud-based services into Tanzu Application Platform. While specific configurations change between services, the overall process remains the same through a consistent set of steps. The aim is to give you enough

understanding so that you can integrate any cloud-based service you want into Tanzu Application Platform.

For a more specific and low-level procedure, see Configure dynamic provisioning of AWS RDS service instances, which provides each step in detail for AWS RDS integration. It might be useful to read through that guide even if you want to integrate with one of the other cloud providers.

About this tutorial

Target user role: Service Operator
Complexity: Advanced
Estimated time: 30 minutes
Topics covered: Dynamic Provisioning, Cloud-based Services, AWS, Azure, GCP, Crossplane
Learning outcomes: An understanding of the steps involved in integrating cloud-based services into Tanzu Application Platform

Concepts

The following is a high-level workflow outlining what is required to integrate a cloud-based service into Tanzu Application Platform.

1. **Install Provider and create ProviderConfig:**
    - Follow the official Upbound documentation to install the Provider and create a ProviderConfig.
2. **Create CompositeResourceDefinition:**
    - Create a CompositeResourceDefinition to define the shape of a new API type representing the service.
    - Choose which (if any) configuration parameters to expose to apps teams.
3. **Create Composition:**
    - Create a Composition using managed resources supplied by the Provider.
    - You can compose as many or as few managed resources as required to generate a service instance that application workloads can connect to and use over the network.
    - (Optional but recommended) Configure the connection secret to adhere to the Service Binding Specification for Kubernetes.
4. **Create provisioner-based ClusterInstanceClass:**
    - Create a provisioner-based ClusterInstanceClass pointing to the CompositeResourceDefinition created earlier.
5. **Create required RBAC:**
    - Create RBAC using the `claim` verb pointing to the provisioner-based ClusterInstanceClass to permit claiming from the class.
6. **Create ClassClaim:**
    - Create a ClassClaim pointing to the provisioner-based ClusterInstanceClass to begin a dynamic provisioning request.
    - Wait for the ClassClaim to report `READY=True`.

Procedure

This tutorial provides the steps required to integrate cloud services, and includes tips and references to example configurations where appropriate.

Step 1: Install a Provider

Install a suitable Crossplane `Provider` for your cloud of choice. Upbound provides support for the three main cloud providers:

- provider-aws
- provider-azure
- provider-gcp

Note

These cloud-based Providers often install many hundreds of additional CRDs onto the cluster, which can have a negative impact on cluster performance. For more information, see Cluster performance degradation due to large number of CRDs.

Choose the Provider you want, and then follow Upbound's official documentation to install the `Provider` and to create a corresponding `ProviderConfig`.

Important

The official documentation for the `Provider` includes a step to "Install Universal Crossplane". You can skip this step because Crossplane is already installed as part of Tanzu Application Platform.

The documentation also assumes Crossplane is installed in the `upbound-system` namespace. However, when working with Crossplane on Tanzu Application Platform, it is installed to the `crossplane-system` namespace by default. Ensure that you use the correct namespace when you create the `Secret` and the `ProviderConfig` with credentials for the `Provider`.

Step 2: Create a CompositeResourceDefinition

Create a `CompositeResourceDefinition`, which defines the shape of a new API type which is used to create the cloud-based resources.

For help creating the `CompositeResourceDefinition`, see the Crossplane documentation, or see Create a CompositeResourceDefinition in *Configure dynamic provisioning of AWS RDS service instances*.

Step 3: Create a Composition

This step is likely to be the most time-consuming. The `Composition` is where you define the configuration for the resources that make up the service instances for app teams to claim. Configure the necessary resources for usable service instances that users can connect to and use over the network.

To get started with creating a `Composition`, first read through Configuring Composition in the Upbound documentation.

You can also see the following `Composition` examples:

- For AWS RDS, see Define composite resource types (AWS).

- For Azure Flexible Server, see Define Composite Resource Types (Azure).
- For GCP Cloud SQL, see Define Composite Resource Types (GCP).

Step 4: Create a provisioner-based ClusterInstanceClass

Create a provisioner-based `ClusterInstanceClass` which is configured to refer to the `CompositeResourceDefinition` created earlier. For example:

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: cloud-service-foo
spec:
  description:
    short: FooDB by cloud provider Foo!
  provisioner:
    crossplane:
      compositeResourceDefinition: NAME-OF-THE-COMPOSITE-RESOURCE-DEFINITION
```

For a real-world example, see Make the service discoverable in *Configure dynamic provisioning of AWS RDS service instances*.

Step 5: Configure RBAC

Create an Role-Based Access Control (RBAC) rule using the `claim` verb pointing to the `ClusterInstanceClass` you created. For example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-foo-db
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups:
  - "services.apps.tanzu.vmware.com"
  resources:
  - clusterinstanceclasses
  resourceNames:
  - cloud-service-foo
  verbs:
  - claim
```

For a real-world example, see Configure RBAC in *Configure dynamic provisioning of AWS RDS service instances*.

Step 6: Verify your integration

To test your integration, create a `ClassClaim` that points to the `ClusterInstanceClass` you created. For example:

```
---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: claim-1
spec:
  classRef:
    name: cloud-service-foo
```

```
  parameters:
    key: value
```

Verify that the `ClassClaim` eventually transitions into a `READY=True` state. If it doesn't, debug the `ClassClaim` using kubectl. For how to do this, see Troubleshoot Services Toolkit.

Abstracting service implementations behind a class across clusters

In this Services Toolkit tutorial you learn how service operators can configure a class that allows for claims to resolve to different backing implementations of a service, such as PostgreSQL, depending on which cluster the class is claimed in.

This sort of setup allows the configurations of workloads and class claims to remain unchanged as they are promoted through environments, whilst also enabling service operators to change the implementations of the backing services without further configuration.

About this tutorial

Target user role: Service Operator
Complexity: Medium
Estimated time: 60 minutes
Topics covered: Classes, Claims, Claim-by-Class, Multi-Cluster
Learning outcomes: Ability to abstract the implementation (for example, helm, tanzu data service, cloud) of a service (for example, RabbitMQ) across multiple clusters

Prerequisites

- Access to three separate Tanzu Application Platform clusters v1.5.0 or later. This tutorial refers to them as `iterate`, `run-test`, and `run-production`, but you can use different names if required.

Scenario

The tutorial is centered around the following hypothetical, but somewhat realistic, real-world scenario.

You work at BigCorp as a service operator. BigCorp uses three separate Tanzu Application Platform clusters: `iterate`, `run-test`, and `run-production`. Application workloads begin on the `iterate` cluster, before being promoted to the `run-test` cluster, and then finally to the `run-production` cluster. The application development team have asked you for a PostgreSQL service they can use with their workloads, which must be available on all three clusters.

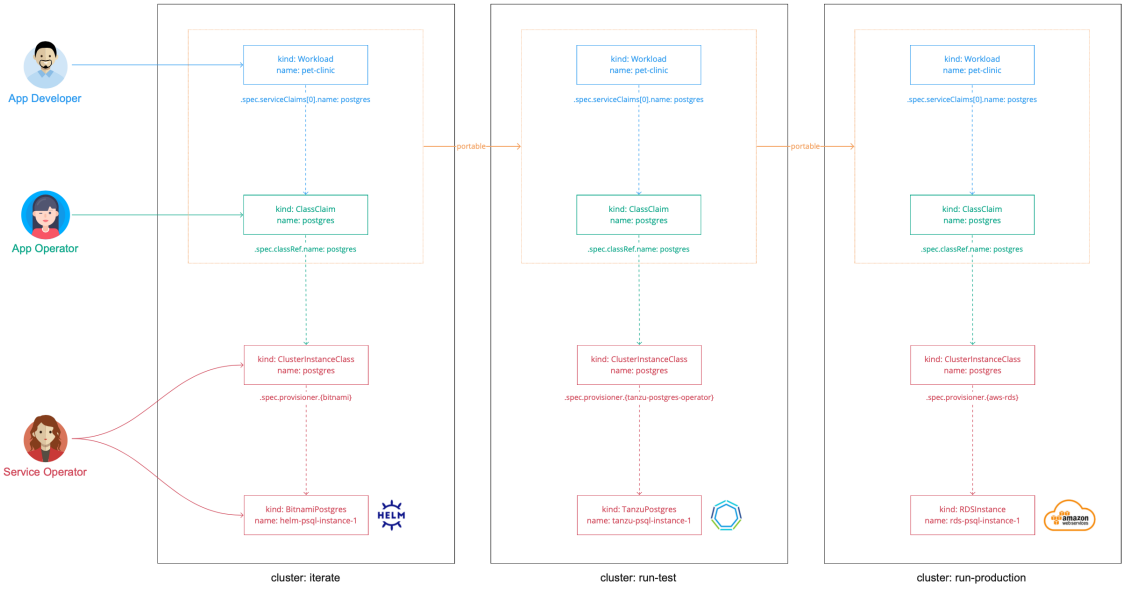
You are aware that the service level objectives (SLOs) for each cluster are different and want to tailor the implementation of the PostgreSQL service to each of the clusters accordingly. The `iterate` cluster has low level SLOs, so you want to offer an unmanaged PostgreSQL service backed by simple Helm chart. The `run-test` cluster has more robust requirements, so want to offer a PostgreSQL service backed by VMware SQL with Postgres for Kubernetes. The `run-production` cluster is critically important, so you want to use a fully managed, cloud-based PostgreSQL implementation there.

You want to ensure that the differing implementations are completely opaque to development teams. They do not need to know about the inner workings of the services, and must be able to keep their workloads and class claims the same as they are promoted across clusters. You have

heard great things about Tanzu Application Platform's claims and classes abstractions and want to make use of them to help you complete your task.

Concepts

This section provides a high-level overview of the elements you will use during this tutorial and how they all fit together.



In this diagram:

- There are three clusters: `iterate`, `run-test`, and `run-production`.
- In each cluster, the service operator creates a `ClusterInstanceClass` called postgres.
    - In the `iterate` cluster, this is a provisioner-based class that uses Bitnami Services to provision Helm instances of PostgreSQL.
    - In the `run-test` cluster, this is a provisioner-based class that uses VMware SQL with Postgres for Kubernetes to provision instances of PostgreSQL.
    - In the `run-production` cluster, this is a provisioner-based class that uses Amazon RDS to provision instances running in Amazon AWS RDS.
- The app operator creates a `ClassClaim`. This is applied with a consuming workload.
    - When it is applied in `iterate` it resolves to a Helm chart instance.
    - When it is promoted to `run-test` it resolves to a VMware PostgreSQL instance.
    - When it is promoted to `run-production` it resolves to an Amazon AWS RDS instance.
- The definition of the `ClassClaim` remains identical across the clusters, which is easier for the application development team.

Important

The backing service implementations and environment layouts used in this scenario are arbitrary. They are not recommendations or requirements.

Although this tutorial uses provisioner-based classes on all three clusters, you can also use a combination of provisioner-based and pool-based classes across the clusters. You might want to do this in cases where, for example, you want to allow for dynamic provisioning of service instances in

the `iterate` cluster, but want to be more considered about the approach in the `run-production` cluster where you might want to ensure that workloads only ever connect to one specific service instance. You can achieve this by using a provisioner-based class on the `iterate` cluster, and an identically named pool-based class on the `run-production` cluster that is configured to only ever select from a pool that consists of one service instance.

Procedure

The following steps explain how to set up a class that allows for claims to resolve to differing implementations of PostgreSQL depending on the cluster it is in.

Step 1: Set up the run-test cluster

Configure the `run-test` cluster for dynamic provisioning of VMware PostgreSQL service instances. To do that, see Configure dynamic provisioning of VMware SQL with Postgres for Kubernetes service instances and complete the steps in the following sections only:

1. Install the Tanzu VMware Postgres Operator
2. Set up the namespace
3. Create a CompositeResourceDefinition
4. Create a Composition
5. Configure RBAC

You do not have to do any other sections in that topic.

Step 2: Set up the run-production cluster

Configure the `run-production` cluster for dynamic provisioning of AWS RDS PostgreSQL service instances. To do that, see Configure Dynamic Provisioning of AWS RDS Service Instances and complete the steps in the following sections only:

1. Install the AWS Provider for Crossplane
2. Create a CompositeResourceDefinition
3. Create a Composition
4. Configure RBAC

You do not have to do any other sections in that topic.

Step 3: Create the class

The `ClusterInstanceClass` acts as the abstraction fronting the differing service implementations across the different clusters. You must create a class with the same name on all three of the clusters, but the configuration of the class varies slightly on each. The `ClassClaim` refers to classes by name. The fact that the class name remains consistent is what allows for the `ClassClaim`, which the application development teams create, to remain unchanged as they are promoted across the clusters.

Create a file named `postgres.class.iterate-cluster.yaml` and copy in the following contents.

```
# postgres.class.iterate-cluster.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: bigcorp-postgresql
```

```
spec:
  description:
    short: PostgreSQL by BigCorp
  provisioner:
    crossplane:
      compositeResourceDefinition: xpostgresqlinstances.bitnami.database.tanzu.vmware.
com
```

This class refers to the `xpostgresqlinstances.bitnami.database.tanzu.vmware.com` CompositeResourceDefinition. This is installed as part of the Bitnami Services package and powers the PostgreSQL service.

You are reusing the underlying CompositeResourceDefinition here from a different class using the class name you want.

Use kubectl to apply the file to the `iterate` cluster.

```
kubectl apply -f postgres.class.iterate-cluster.yaml
```

Create a file named `postgres.class.run-test-cluster.yaml` and copy in the following contents.

```
# postgres.class.run-test-cluster.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: bigcorp-postgresql
spec:
  description:
    short: PostgreSQL by BigCorp
  provisioner:
    crossplane:
      compositeResourceDefinition: xpostgresqlinstances.database.tanzu.example.org
```

This class is almost identical to the previous one, however this one refers instead to the `xpostgresqlinstances.database.tanzu.example.org` CompositeResourceDefinition.

Use kubectl to apply the file to the `run-test` cluster.

```
kubectl apply -f postgres.class.run-test-cluster.yaml
```

Create a file named `postgres.class.run-production-cluster.yaml` and copy in the following contents.

```
# postgres.class.run-production-cluster.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: bigcorp-postgresql
spec:
  description:
    short: PostgreSQL by BigCorp
  provisioner:
    crossplane:
      compositeResourceDefinition: xpostgresqlinstances.database.rds.example.org
```

Again, this class is almost identical to the previous two, but this time refers to the `xpostgresqlinstances.database.rds.example.org` CompositeResourceDefinition.

Use kubectl to apply the file to the `run-production` cluster.

```
kubectl apply -f postgres.class.run-production-cluster.yaml
```

Step 4: Create and promote the workload and class claim

After configuring the clusters and classes, switch roles from service operator to application operator and developer to create the workload and class claim YAML and promote it through the three clusters.

Create a file named `app-with-postgres.yaml` and copy in the following contents.

```
# app-with-postgres.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim
metadata:
  name: postgres
  namespace: default
spec:
  classRef:
    name: bigcorp-postgresql

---
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: pet-clinic
  namespace: default
  labels:
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: pet-clinic
spec:
  params:
  - name: annotations
    value:
      autoscaling.knative.dev/minScale: "1"
  env:
  - name: SPRING_PROFILES_ACTIVE
    value: postgres
  serviceClaims:
  - name: db
    ref:
      apiVersion: services.apps.tanzu.vmware.com/v1alpha1
      kind: ClassClaim
      name: postgres
  source:
    git:
      url: https://github.com/sample-accelerators/spring-petclinic
      ref:
        branch: main
        tag: tap-1.2
```

Then use kubectl to apply the file to the `iterate` cluster.

```
kubectl apply -f app-with-postgres.yaml
```

Wait for the workload to become ready and then inspect the cluster to see that the workload is bound to a Helm-based PostgreSQL service instance. Target the `iterate` cluster then run `helm list -A` to confirm.

Next, apply the exact same `app-with-postgres.yaml` to the `run-test` cluster. When it is ready, confirm that the workload is bound to a Tanzu-based PostgreSQL service instance. Target the `run-test` cluster then run `kubectl get postgres -n tanzu-psql-service-instances` to confirm.

Finally, apply the exact same `app-with-postgres.yaml` to the `run-production` cluster. When it is ready, confirm that the workload is bound to a RDS-based PostgreSQL service instance. Target the `run-production` cluster then run `kubectl get RDSInstance -A` to confirm.

Using direct secret references

In this Services Toolkit tutorial you learn how developers can use direct references to Kubernetes `Secret` resources to connect their application workloads to almost any backing service.

This includes backing services that:

- Run external to Tanzu Application Platform
- Do not adhere to ProvisionedService in the Service Binding Specification for Kubernetes in GitHub.

If you are familiar with Cloud Foundry and Tanzu Application Service, this capability is similar to the concept of user-provided service instances. For more information about user-provided service instances in Cloud Foundry, see the Cloud Foundry documentation.

This tutorial demonstrates a procedure to bind a new application on Tanzu Application Platform to an existing PostgreSQL database that exists in Azure. However, the steps are applicable to any backing service that you want to connect to.

About this tutorial

Target user role: Service Operator and Application Operator
Complexity: Easy
Estimated time: 10 minutes
Topics covered: Service Binding, Direct Secret References
Learning outcomes: Ability to bind workloads to almost any backing service using direct secret references

Prerequisites

Before you can follow this tutorial, you must have:

- Access to a Tanzu Application Platform cluster v1.5.0 or later.
- An Azure PostgreSQL database to connect to.
- Configured networking between the workload and the service endpoint and you must have the credentials for the backing service. Whether this requires extra steps depends on your Kubernetes distribution and the backing service you want to connect your Tanzu Application Platform workloads to.

Create a binding-compatible secret

1. Create a file named `external-azure-db-binding-compatible.yaml` and enter a Kubernetes secret resource similar to the following example:

```
# external-azure-db-binding-compatible.yaml
---
apiVersion: v1
kind: Secret
metadata:
  name: external-azure-db-binding-compatible
type: Opaque
stringData:
```

```
  type: postgresql
  provider: azure
  host: EXAMPLE.DATABASE.AZURE.COM
  port: "5432"
  database: "EXAMPLE-DB-NAME"
  username: "USER@EXAMPLE"
  password: "PASSWORD"
```

Substitute in the values as required.

When using direct secret references, the `Secret` values must abide by the Well-known Secret Entries specifications as defined by the Service Binding Specification for Kubernetes. If you plan to bind this secret to a Spring-based application workload and want to take advantage of the auto-wiring feature, this secret must also contain the properties required by Spring Cloud Bindings.

2. Apply the YAML file by running:

```
kubectl apply -f external-azure-db-binding-compatible.yaml
```

If you are using a multicluster Tanzu Application Platform topology, apply the YAML file to all Run clusters.

3. In a file named `stk-secret-reader.yaml`, grant sufficient Role-Based Access Control (RBAC) permissions to permit Services Toolkit to read the secrets specified by the class:

```
# stk-secret-reader.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: stk-secret-reader
  labels:
    servicebinding.io/controller: "true"
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
  - list
  - watch
```

4. Apply your changes by running:

```
kubectl apply -f stk-secret-reader.yaml
```

If you are using a multicluster Tanzu Application Platform topology, apply the YAML file to all Run clusters.

5. Create a claim for the newly created secret by running:

```
tanzu service resource-claim create external-azure-db-claim \
  --resource-name external-azure-db-binding-compatible \
  --resource-kind Secret \
  --resource-api-version v1
```

If you are using a multicluster Tanzu Application Platform topology, create the claim on the Build cluster.

6. Obtain the claim reference of the claim by running:

```
tanzu service resource-claim list -o wide
```

If you are using a multicluster Tanzu Application Platform topology, obtain the claim reference on the Build cluster.

Expected output:

```
NAME                     READY  REASON  CLAIM REF
external-azure-db-claim  True           services.apps.tanzu.vmware.com/v1alpha
1:ResourceClaim:external-azure-db-claim
```

From the output, record the value of `CLAIM REF`.

7. Create an application workload by running a command similar to the following example:

```
tanzu apps workload create WORKLOAD-NAME \
  --git-repo https://github.com/sample-accelerators/spring-petclinic \
  --git-branch main \
  --git-tag tap-1.2 \
  --type web \
  --label app.kubernetes.io/part-of=spring-petclinic \
  --annotation autoscaling.knative.dev/minScale=1 \
  --env SPRING_PROFILES_ACTIVE=postgres \
  --service-ref db=REFERENCE
```

Where:

- `WORKLOAD-NAME` is the name of the application workload. For example, `pet-clinic`.
- `REFERENCE` is the value of the `CLAIM REF` for the newly created claim in the output of the last step.

If you are using a multicluster Tanzu Application Platform topology, create the application workload on the Build cluster.

Services Toolkit how-to guides

This section contains how-to guides for Services Toolkit.

In this section:

- Authorize users and groups to claim from provisioner-based classes
- Configure dynamic provisioning of AWS RDS service instances
- Configure dynamic provisioning of VMware SQL with Postgres for Kubernetes service instances
- Configure private registry and VMware Tanzu Application Catalog integration for Bitnami Services
- Troubleshooting and known limitations

Authorize users and groups to claim from provisioner-based classes

This Services Toolkit topic for service operators explains how you configure access control so that the required users and groups have authorization to claim from provisioner-based classes.

By default, only users with `cluster-admin` privileges are authorized to create claims for provisioner-based classes. This is because creating claims for provisioner-based classes creates new service

instances, all of which consume resources and might incur monetary cost. As such, you might want to configure some form of access control.

There is one exception to this rule, which is that by default, users with the `app-operator` user role are authorized to create claims for the provisioner-based classes that are part of the Bitnami Services package. For how-to deactivate this default behavior, see Revoke default authorization for claiming from the Bitnami Services classes later in this topic.

Access control is implemented through standard Kubernetes Role-Based Access Control (RBAC) with the use of the custom verb `claim`. You must create a rule in a `ClusterRole` which specifies the `claim` verb for one or more `clusterinstanceclasses`, and then bind the `ClusterRole` to the roles that you want to authorize to create claims for classes with a `ClusterRoleBinding`. This approach is particularly effective when paired with Tanzu Application Platform's aggregated user roles. For more information about user roles in Tanzu Application Platform, see Role descriptions.

Authorize all users with the app-operator user role to claim from any namespace

Create a `ClusterRole` with a rule that specifies the `claim` verb for one or more `ClusterInstanceClass` resources and apply the relevant label.

For example:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-class-bigcorp-rabbitmq
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups:
  - services.apps.tanzu.vmware.com
  resources:
  - clusterinstanceclasses
  resourceNames:
  - bigcorp-rabbitmq
  verbs:
  - claim
```

This example specifies a `ClusterRole` that permits claiming from a class named `bigcorp-rabbitmq`. The example also includes the `apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"` label, which causes this `ClusterRole` to aggregate to Tanzu Application Platform's app-operator user role at the cluster scope.

The result is that any user who has the `app-operator` role is now authorized to claim from the `bigcorp-rabbitmq` class. By default, the `app-operator` user role is authorized to create claims for the provisioner-based class.

Authorize a user to claim from a specific namespace

Create a `ClusterRole` with a rule that specifies the `claim` verb for one or more `ClusterInstanceClass` resource and a corresponding `RoleBinding` to bind it to a user.

For example:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  name: claim-class-bigcorp-rabbitmq
rules:
- apiGroups:
  - services.apps.tanzu.vmware.com
  resources:
  - clusterinstanceclasses
  resourceNames:
  - bigcorp-rabbitmq
  verbs:
  - claim

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: alice-claim-class-bigcorp-rabbitmq
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: claim-class-bigcorp-rabbitmq
subjects:
- kind: User
  name: "alice@example.com"
  apiGroup: rbac.authorization.k8s.io
```

This example specifies a `ClusterRole` that permits claiming from a class named `bigcorp-rabbitmq`. The YAML also creates a `ClusterRoleBinding`that binds the user `alice@example.com` to the `ClusterRole`.

The result is that `alice@example.com` is now authorized to claim from `bigcorp-rabbitmq` class.

The user `alice@example.com` still needs permission to create `ClassClaims` in namespaces that they want to consume the services from.

The following example gives `alice@example.com` permission to get, create, update, or delete `ClassClaims` in the `apps` namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
   name: create-class-claim-example
   namespace: apps
rules:
  - apiGroups:
    - services.apps.tanzu.vmware.com
    resources:
    - classclaims
    verbs:
    - get
    - create
    - update
    - delete
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rbac-role-binding-role-binding
  namespace: apps
subjects:
  - kind: User
    name: "alice@example.com"
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
```

```
  name: create-class-claim-example
  apiGroup: rbac.authorization.k8s.io
```

Revoke default authorization for claiming from the Bitnami Services classes

By default, users with the `app-operator` user role are authorized to create claims for the provisioner-based classes which are part of the Bitnami Services package.

To revoke this authorization:

1. Add the following to your `tap-values.yaml` file:

   ```
   bitnami_services:
     globals:
       create_clusterroles: false
   ```

2. Update Tanzu Application Platform by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com --values-file tap-va
   lues.yaml -n tap-install
   ```

The result is that any user who has the `app-operator` role is now not authorized to create claims for any of the Bitnami services in any namespace on the cluster.

Configure dynamic provisioning of AWS RDS service instances

This Services Toolkit topic tells you how service operators can set up dynamic provisioning. This enables app development teams to create self-serve AWS RDS service instances that are customized to meet their needs.

If you are not already familiar with dynamic provisioning in Tanzu Application Platform, following the tutorial Set up dynamic provisioning of service instances might help you to understand the steps presented in this topic.

Prerequisites

Before you configure dynamic provisioning, you must have:

- Access to a Tanzu Application Platform cluster v1.5.0 or later.
- The Tanzu services CLI plug-in v0.6.0 or later.
- Access to AWS.

Configure dynamic provisioning

To configure dynamic provisioning for AWS RDS service instances, you must:

1. Install the AWS Provider for Crossplane
2. Create a CompositeResourceDefinition
3. Create a Composition
4. Make the service discoverable
5. Configure RBAC
6. Verify your configuration

Install the AWS Provider for Crossplane

The first step is to install the AWS `Provider` for Crossplane.

There are two variants of the `Provider`:

- crossplane-contrib/provider-aws
- upbound/provider-aws

VMware recommends that you install the official Upbound variant. To install the `Provider` and to create a corresponding `ProviderConfig`, see the Upbound documentation.

Important

The official documentation for the `Provider` includes a step to “Install Universal Crossplane”. You can skip this step because Crossplane is already installed as part of Tanzu Application Platform.

The documentation also assumes Crossplane is installed in the `upbound-system` namespace. However, when working with Crossplane on Tanzu Application Platform, it is installed to the `crossplane-system` namespace by default. Ensure that you use the correct namespace when you create the `Secret` and the `ProviderConfig` with credentials for the `Provider`.

Create a CompositeResourceDefinition

To create the CompositeResourceDefinition (XRD):

1. Create a file named `xpostgresqlinstances.database.rds.example.org.xrd.yaml` and copy in the following contents:

```
# xpostgresqlinstances.database.rds.example.org.xrd.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: CompositeResourceDefinition
metadata:
  name: xpostgresqlinstances.database.rds.example.org
spec:
  claimNames:
    kind: PostgreSQLInstance
    plural: postgresqlinstances
  connectionSecretKeys:
  - type
  - provider
  - host
  - port
  - database
  - username
  - password
  group: database.rds.example.org
  names:
    kind: XPostgreSQLInstance
    plural: xpostgresqlinstances
  versions:
  - name: v1alpha1
    referenceable: true
    schema:
      openAPIV3Schema:
        properties:
          spec:
            properties:
```

```
              storageGB:
                type: integer
                default: 20
            type: object
        type: object
    served: true
```

This XRD configures the parameter `storageGB`. This gives application teams the option to choose a suitable amount of storage for the AWS RDS service instance when they create a claim. You can choose to expose as many or as few parameters to application teams as you like.

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f xpostgresqlinstances.database.rds.example.org.xrd.yaml
```

Create a Composition

To create the composition:

1. Create a file named `xpostgresqlinstances.database.rds.example.org.composition.yaml` and copy in the following contents:

```
# xpostgresqlinstances.database.rds.example.org.composition.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  labels:
    provider: "aws"
    vpc: "default"
  name: xpostgresqlinstances.database.rds.example.org
spec:
  compositeTypeRef:
    apiVersion: database.rds.example.org/v1alpha1
    kind: XPostgreSQLInstance
  publishConnectionDetailsWithStoreConfigRef:
    name: default
  resources:
  - base:
      apiVersion: database.aws.crossplane.io/v1beta1
      kind: RDSInstance
      spec:
        forProvider:
          # NOTE: configure this section to your specific requirements
          dbInstanceClass: db.t2.micro
          engine: postgres
          dbName: postgres
          engineVersion: "12"                    # <---- Refer to https://doc
s.aws.amazon.com/AmazonRDS/latest/PostgreSQLReleaseNotes/postgresql-release-cal
endar.html for latest
          masterUsername: masteruser
          publiclyAccessible: true               # <---- DANGER
          region: us-east-1
          skipFinalSnapshotBeforeDeletion: true
        writeConnectionSecretToRef:
          namespace: crossplane-system
    connectionDetails:
    - name: type
      value: postgresql
    - name: provider
      value: aws
    - name: database
```

```
      value: postgres
    - fromConnectionSecretKey: username
    - fromConnectionSecretKey: password
    - name: host
      fromConnectionSecretKey: endpoint
    - fromConnectionSecretKey: port
    name: rdsinstance
    patches:
    - fromFieldPath: metadata.uid
      toFieldPath: spec.writeConnectionSecretToRef.name
      transforms:
      - string:
          fmt: '%s-postgresql'
          type: Format
        type: string
      type: FromCompositeFieldPath
    - fromFieldPath: spec.storageGB
      toFieldPath: spec.forProvider.allocatedStorage
      type: FromCompositeFieldPath
```

2. Configure the Composition you just copied to your specific requirements.

   In particular, you can deactivate the `publiclyAccessible: true` setting. When set to `true`, this setting opens up public access to all dynamically provisioned RDS databases. When set to `false`, only internal connectivity is allowed.

   To help you configure the Composition, see this example in the Upbound documentation. The example defines a composition that creates a separate VPC for each RDS PostgreSQL instance and automatically configures inbound rules.

3. Apply the file to the Tanzu Application Platform cluster by running:

   ```
   kubectl apply -f xpostgresqlinstances.database.rds.example.org.composition.yaml
   ```

Make the service discoverable

To make the service discoverable to application teams:

1. Create a file named `rds.class.yaml` and copy in the following contents:

   ```
   # rds.class.yaml

   ---
   apiVersion: services.apps.tanzu.vmware.com/v1alpha1
   kind: ClusterInstanceClass
   metadata:
     name: aws-rds-psql
   spec:
     description:
       short: Amazon AWS RDS PostgreSQL
     provisioner:
       crossplane:
         compositeResourceDefinition: xpostgresqlinstances.database.rds.example.or
   g
   ```

2. Apply the file to the Tanzu Application Platform cluster by running:

   ```
   kubectl apply -f rds.class.yaml
   ```

Configure RBAC

To configure Role-Based Access Control (RBAC) to authorize users with the app-operator role to claim from the class:

1. Create a file named `app-operator-claim-aws-rds-psql.rbac.yaml` and copy in the following contents:

```
# app-operator-claim-aws-rds-psql.rbac.yaml

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-aws-rds-psql
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups:
  - "services.apps.tanzu.vmware.com"
  resources:
  - clusterinstanceclasses
  resourceNames:
  - aws-rds-psql
  verbs:
  - claim
```

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f app-operator-claim-aws-rds-psql.rbac.yaml
```

Verify your configuration

To verify your configuration, create a claim for an AWS RDS service instance by running:

```
tanzu service class-claim create rds-psql-1 --class aws-rds-psql -p storageGB=30
```

Note

Whether application workloads can establish network connectivity to the resulting RDS database depends on a number of factors. This includes specifics about the environment you're working in and the configuration in the `Composition` file. At a minimum, you can configure a `securityGroup` to permit inbound traffic. There might be other requirements as well.

Configure dynamic provisioning of VMware SQL with Postgres for Kubernetes service instances

This Services Toolkit topic for service operators explains how you set up dynamic provisioning. This enables app development teams to create self-serve VMware SQL with Postgres for Kubernetes service instances that are customized to meet their needs.

If you are not already familiar with dynamic provisioning in Tanzu Application Platform, following the tutorial Set up dynamic provisioning of service instances. might be help you understand the steps presented in this topic.

Prerequisites

Before you configure dynamic provisioning, you must have:

- Access to a Tanzu Application Platform cluster v1.5.0 or later.
- The Tanzu services CLI plug-in v0.6.0 or later.

Configure dynamic provisioning

To configure dynamic provisioning for VMware SQL with Postgres for Kubernetes services instances, you must:

1. Install the VMware Postgres Operator
2. Set up the namespace
3. Create a CompositeResourceDefinition
4. Create a Composition
5. Make the service discoverable
6. Configure RBAC
7. Verify your configuration

Install the VMware Postgres Operator

Install the VMware Postgres Operator by following the steps in Installing a VMware Postgres Operator.

Set up the namespace

This topic configures dynamic provisioning to provision all PostgreSQL service instances into the same namespace. This namespace is named `tanzu-psql-service-instances`.

To set up the namespace:

1. Ensure that the namespace exists by running the following:

   ```
   kubectl create namespace tanzu-psql-service-instances
   ```

2. The VMware Postgres Operator also requires that a secret holding registry credentials exists in the same namespace that the service instances will be created in. Ensure that the secret exists in the namespace by running:

   ```
   kubectl create secret --namespace=tanzu-psql-service-instances docker-registry
   regsecret \
     --docker-server=https://registry.tanzu.vmware.com \
     --docker-username=`USERNAME` \
     --docker-password=`PASSWORD`
   ```

   Where:

   - `USERNAME` is your registry username.
   - `PASSWORD` is your registry password.

   Note

   You must update the `--docker-server` value if you relocated images as part of the installation of the operator.

Create a CompositeResourceDefinition

To create the CompositeResourceDefinition (XRD):

1. Create a file named `xpostgresqlinstances.database.tanzu.example.org.xrd.yaml` and copy in the following contents:

```
# xpostgresqlinstances.database.tanzu.example.org.xrd.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: CompositeResourceDefinition
metadata:
  name: xpostgresqlinstances.database.tanzu.example.org
spec:
  connectionSecretKeys:
  - provider
  - type
  - database
  - host
  - password
  - port
  - uri
  - username
  group: database.tanzu.example.org
  names:
    kind: XPostgreSQLInstance
    plural: xpostgresqlinstances
  versions:
  - name: v1alpha1
    referenceable: true
    schema:
      openAPIV3Schema:
        properties:
          spec:
            properties:
              storageGB:
                type: integer
                default: 20
            type: object
        type: object
    served: true
```

This XRD configures the parameter `storageGB`. This gives application teams the option to choose a suitable amount of storage for the Tanzu Postgres service instance when they create a claim. You can choose to expose as many or as few parameters to application teams as you like.

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f xpostgresqlinstances.database.tanzu.example.org.xrd.yaml
```

Create a Composition

To create the Composition:

1. Create a file named `xpostgresqlinstances.database.tanzu.example.org.composition.yaml` and copy in the following contents:

```
# xpostgresqlinstances.database.tanzu.example.org.composition.yaml

---
apiVersion: apiextensions.crossplane.io/v1
kind: Composition
metadata:
  name: xpostgresqlinstances.database.tanzu.example.org
spec:
  compositeTypeRef:
    apiVersion: database.tanzu.example.org/v1alpha1
    kind: XPostgreSQLInstance
```

```
publishConnectionDetailsWithStoreConfigRef:
  name: default
resources:
- base:
    apiVersion: kubernetes.crossplane.io/v1alpha1
    kind: Object
    spec:
      forProvider:
        manifest:
          apiVersion: sql.tanzu.vmware.com/v1
          kind: Postgres
          metadata:
            name: PATCHED
            namespace: tanzu-psql-service-instances
          spec:
            storageSize: 2G
      connectionDetails:
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.provider
        toConnectionSecretKey: provider
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.type
        toConnectionSecretKey: type
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.host
        toConnectionSecretKey: host
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.port
        toConnectionSecretKey: port
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.username
        toConnectionSecretKey: username
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.password
        toConnectionSecretKey: password
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.database
        toConnectionSecretKey: database
      - apiVersion: v1
        kind: Secret
        namespace: tanzu-psql-service-instances
        fieldPath: data.uri
        toConnectionSecretKey: uri
      writeConnectionSecretToRef:
        namespace: tanzu-psql-service-instances
  connectionDetails:
  - fromConnectionSecretKey: provider
  - fromConnectionSecretKey: type
  - fromConnectionSecretKey: host
  - fromConnectionSecretKey: port
  - fromConnectionSecretKey: username
  - fromConnectionSecretKey: password
```

```
    - fromConnectionSecretKey: database
    - fromConnectionSecretKey: uri
    patches:
      - fromFieldPath: metadata.name
        toFieldPath: spec.forProvider.manifest.metadata.name
        type: FromCompositeFieldPath
      - fromFieldPath: spec.storageSize
        toFieldPath: spec.forProvider.manifest.spec.persistence.storage
        transforms:
        - string:
            fmt: '%dG'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.writeConnectionSecretToRef.name
        transforms:
        - string:
            fmt: '%s-psql'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[0].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[1].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[2].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[3].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[4].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[5].name
        transforms:
        - string:
```

```
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[6].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
      - fromFieldPath: metadata.name
        toFieldPath: spec.connectionDetails[7].name
        transforms:
        - string:
            fmt: '%s-app-user-db-secret'
            type: Format
          type: string
        type: FromCompositeFieldPath
    readinessChecks:
      - type: MatchString
        fieldPath: status.atProvider.manifest.status.currentState
        matchString: "Running"
```

2. Configure the Composition you just copied to your specific requirements.
3. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f xpostgresqlinstances.database.tanzu.example.org.composition.ya
ml
```

Make the service discoverable

To make the service discoverable to application teams:

1. Create a file named `tanzu-psql.class.yaml` and copy in the following contents:

```
# tanzu-psql.class.yaml

---
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass
metadata:
  name: tanzu-psql
spec:
  description:
    short: VMware SQL with Postgres
  provisioner:
    crossplane:
      compositeResourceDefinition: xpostgresqlinstances.database.tanzu.example.
org
```

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f tanzu-psql.class.yaml
```

Configure RBAC

To configure access control with RBAC:

1. Create a file named `provider-kubernetes-tanzu-postgres-read-writer.rbac.yaml` and copy in the following contents:

```
# provider-kubernetes-tanzu-postgres-read-writer.rbac.yaml

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: tanzu-postgres-read-writer
  labels:
    services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"
rules:
- apiGroups:
  - sql.tanzu.vmware.com
  resources:
  - postgres
  verbs:
  - "*"
```

2. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f provider-kubernetes-tanzu-postgres-read-writer.rbac.yaml
```

3. Create a file named `app-operator-claim-tanzu-psql.rbac.yaml` and copy in the following contents:

```
# app-operator-claim-tanzu-psql.rbac.yaml

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-tanzu-psql
  labels:
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
- apiGroups:
  - "services.apps.tanzu.vmware.com"
  resources:
  - clusterinstanceclasses
  resourceNames:
  - tanzu-psql
  verbs:
  - claim
```

4. Apply the file to the Tanzu Application Platform cluster by running:

```
kubectl apply -f app-operator-claim-tanzu-psql.rbac.yaml
```

Verify your configuration

To verify your configuration, create a claim for a PostgreSQL service instance by running:

```
tanzu service class-claim create tanzu-psql-1 --class tanzu-psql -p storageGB=5
```

Troubleshoot Services Toolkit

This topic explains how you can troubleshoot issues related to working with services on Tanzu Application Platform (commonly known as TAP).

For the limitations of services on Tanzu Application Platform, see Services Toolkit limitations.

Debug `ClassClaim` and provisioner-based `ClusterInstanceClass`

This section provides guidance on how to debug issues related to using `ClassClaim` and provisioner-based `ClusterInstanceClass`. The approach starts by inspecting a `ClassClaim` and tracing back through the chain of resources that are created when fulfilling the `ClassClaim`.

Prerequisites

To follow the steps in this section, you must have kubectl access to the cluster.

Step 1: Inspect the `ClassClaim`, `ClusterInstanceClass`, and `CompositeResourceDefinition`

1. Inspect the status of `ClassClaim` by running:

   ```
   kubectl describe classclaim claim-name -n NAMESPACE
   ```

   Where `NAMESPACE` is your namespace.

   From the output, check the following:

   - Check the status conditions for information that can lead you to the cause of the issue.
   - Check `.spec.classRef.name` and record the value.

2. Inspect the status of the `ClusterInstanceClass` by running:

   ```
   kubectl describe clusterinstanceclass CLASS-NAME
   ```

   Where `CLASS-NAME` is the value of `.spec.classRef.name` you retrieved in the previous step.

   From the output, check the following:

   - Check the status conditions for information that can lead you to the cause of the issue.
   - Check that the `Ready` condition has status `"True"`.
   - Check `.spec.provisioner.crossplane` and record the value.

3. Inspect the status of the `CompositeResourceDefinition` by running:

   ```
   kubectl describe xrd XRD-NAME
   ```

   Where `XRD-NAME` is the value of `.spec.provisioner.crossplane` you retrieved in the previous step.

   From the output, check the following:

   - Check the status conditions for information that can lead you to the cause of the issue.
   - Check that the `Established` condition has status `"True"`.
   - Check events for any errors or warnings that can lead you to the cause of the issue.
   - If both the `ClusterInstanceClass` reports `Ready="True"` and the `CompositeResourceDefinition` reports `Established="True"`, move on to the next section.

Step 2: Inspect the Composite Resource, the Managed Resources and the underlying resources

1. Check `.status.provisionedResourceRef` by running:

   ```
   kubectl describe classclaim claim-name -n NAMESPACE
   ```

   Where `NAMESPACE` is your namespace.

   From the output, check the following:

   - Check `.status.provisionedResourceRef`, and record the values of `kind`, `apiVersion`, and `name`.

2. Inspect the status of the Composite Resource by running:

   ```
   kubectl describe KIND.API-GROUP NAME
   ```

   Where:

   - `KIND` is the value of `kind` you retrieved in the previous step.
   - `API-GROUP` is the value of `apiVersion` you retrieved in the previous step without the `/<version>` part.
   - `NAME` is the value of `name` you retrieved in the previous step.

   From the output, check the following:

   - Check the status conditions for information that can lead you to the cause of the issue.
   - Check that the `Synced` condition has status `"True"`. If it doesn't then there was an issue creating the Managed Resources from which this Composite Resource is composed. Refer to `.spec.resourceRefs` in the output and for each:
     - Use the values of `kind`, `apiVersion`, and `name` to inspect the status of the Managed Resource.
     - Check the status conditions for information that can lead you to the cause of the issue.
   - Check events for any errors or warnings that can lead you to the cause of the issue.
   - If all Managed Resources appear healthy, move on to the next section.

Step 3: Inspect the events log

Inspect the events log by running:

```
kubectl get events -A
```

From the output, check the following:

- Check for any errors or warnings that can lead you to the cause of the issue.
- If there are no errors or warnings, move on to the next section.

Step 4: Inspect the secret

1. Check `.status.resourceRef` by running:

   ```
   kubectl get classclaim claim-name -n NAMESPACE -o yaml
   ```

   Where `NAMESPACE` is your namespace.

From the output, check the following:

- Check `.status.resourceRef` and record the values `kind`, `apiVersion`, `name`, and `namespace`

2. Inspect the claimed resource, which is likely a secret, by running:

```
kubectl get secret NAME -n NAMESPACE -o yaml
```

Where:

- `NAME` is the `name` you retrieved in the previous step.
- `NAMESPACE` is the `namespace` you retrieved in the previous step.

If the secret is there and has data, then something else must be causing the issue.

Step 5: Contact support

If you have followed the steps in this section and are still unable to discover the cause of the issue, contact VMware Support for further guidance and help to resolve the issue.

Unexpected error if `additionalProperties` is `true` in a CompositeResourceDefinition

Symptom:

When creating a `CompositeResourceDefinition`, if you set `additionalProperties: true` in the `openAPIV3Schema` section, an error occurs during the validation step of the creation of any `ClassClaim` that refers to a class that refers to the `CompositeResourceDefinitions`.

The error appears as follows:

```
json: cannot unmarshal bool into Go struct field JSONSchemaProps.AdditionalProperties
of type apiextensions.JSONSchemaPropsOrBool
```

Solution:

Rather than setting `additionalProperties: true`, you can set `additionalProperties: {}`. This has the same effect, but does not cause unexpected errors.

Default cluster-admin IAM roles on GKE do not allow you to claim Bitnami Services

Symptom:

For Tanzu Application Platform installations on Google Kubernetes Engine (GKE) clusters, users with cluster-admin Role-Based Access Control (RBAC) permissions are not able to create class claims for any of the Bitnami Services.

The following error occurs:

```
Error: admission webhook "vclassclaim.validation.resourceclaims.services.apps.tanzu.vm
ware.com" denied the request: user 'user@example.com' cannot 'claim' from clusterinsta
nceclass 'mysql-unmanaged'
Error: exit status 1
```

Solution:

Explicitly create a ClusterRoleBinding for your user or group to the corresponding `app-operator-claim-class-SERVICE` ClusterRole, where `SERVICE` is one of `mysql`, `postgresql`, `rabbitmq`, or `redis`.

Cannot claim from clusterinstanceclass when creating a `ClassClaim`

Symptom:

Users who were previously able to create a `ClassClaim` now get an admission error similar to:

```
user 'alice@example.com' cannot 'claim' from clusterinstanceclass 'bigcorp-rabbitmq'
```

This occurs even if users were granted the `claim` permission on `ClusterInstanceClasses` through either:

- A `Role` and a `RoleBinding`
- A `ClusterRole` and a `RoleBinding`

Explanation:

You now need the cluster-level `claim` permission, granted through a `ClusterRole` and `ClusterRoleBinding`. Namespace-scoped permissions are no longer enough. This is to protect against unexpected access to resources in other namespaces.

This change was introduced with Services Toolkit v0.10.3 in Tanzu Application Platform v1.5.6. For more information about this change, see The claim verb for ClusterInstanceClass.

Solution:

To allow users to create `ClassClaims` again, you must:

1. Move from a `Role` to a `ClusterRole` for granting users permission to claim a `ClusterInstanceClass`.
2. Move from a `RoleBinding` to a `ClusterRoleBinding` for binding this permission to a user.

For more information, see Authorize users and groups to claim from provisioner-based classes.

Services Toolkit reference

This section provides reference documentation for Services Toolkit.

In this section:

- API documentation
- Tanzu Service CLI Plug-In
- Services Toolkit terminology and user roles

Services Toolkit API documentation

This section of the documentation provides detailed information about Services Toolkit's APIs.

- ClusterInstanceClass and ClassClaim
- ResourceClaim and ResourceClaimPolicy
- InstanceQuery
- RBAC

ClusterInstanceClass and ClassClaim

This topic provides Services Toolkit API documentation for `ClusterInstanceClass` and `ClassClaim`.

ClusterInstanceClass

You can configure `ClusterInstanceClass` to one of two variants - either pool-based or provisioner-based.

- Claims for pool-based classes are fulfilled by identifying service instances using the configuration in `.spec.pool`.
- Claims for provisioner-based classes are fulfilled by provisioning new service instances using the configuration in `.spec.provisioner`.

A class can either be a pool-based class or a provisioner-based class, but never both.

The following snippet outlines the `ClusterInstanceClass` YAML:

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClusterInstanceClass

metadata:
  # A name for the class. The class name is used by consumers (application operators
  # and developers) when creating claims.
  name: mysql-unmanaged

spec:
  # Provide information about the class in the description.
  description:
    # A short description for the class. Aim to provide just enough information
    # to help consumers (application operators and developers) to understand
    # what's on offer by the class.
    short: MySQL by Bitnami

  # (Optional) Configure a provisioner-based class.
  # Must specify one of either `provisioner` or `pool`.
  provisioner:
    # Configure provisioning using Crossplane (https://www.crossplane.io/).
    crossplane:
      # CompositeResourceDefinition refers to the name of a Composite Resource Definit
ion (XRD).
      # For example, "xpostgresqlinstances.database.example.org".
      compositeResourceDefinition: xmysqlinstances.bitnami.database.tanzu.vmware.com

      # (Optional) The compositionSelector allows you to match a Composition by
      # labels rather than naming one explicitly. It is used to set the compositionRef
      # if none is specified explicitly.
      compositionSelector:
        matchLabels:
          provider: bitnami
          type: mysql

      # (Optional) CompositionRef specifies which Composition this XR will use to
      # compose resources when it is created, updated, or deleted.
      # This can be omitted and is set automatically if the XRD has a default or
      # enforced composition reference, or if the below composition selector is set.
      compositionRef:
        name: composition-name

      # (Optional) CompositionUpdatePolicy specifies how existing XRs should be
      # updated to new revisions of their underlying composition.
      # One of either 'Automatic' or 'Manual'; default=Automatic.
      compositionUpdatePolicy: Manual

  # (Optional) Configure a pool-based class.
  # Must specify one of either `provisioner` or `pool`.
  pool:
    # (Optional) Group specifies the API group for the resources belonging to this cla
```

```
ss.
    group:

    # Kind specifies the API Kind for the resources belonging to this class.
    kind:

    # (Optional) FieldSelector specifies a set of fields that MUST match certain condi
tions.
    # See https://kubernetes.io/docs/concepts/overview/working-with-objects/field-sele
ctors/.
    fieldSelector:

    # (Optional) LabelSelector specifies a set of labels that MUST match.
    # See https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#la
bel-selectors.
    labelSelector:

# status is populated by the controller
status:
  # Conditions for the class.
  conditions:
    # The condition type. Currently only 'Ready'.
    - type: Ready
      # status can be either 'True' or 'False'.
      status: "True"
      # reason provides a reason for status: "False" for additional context.
      # One of 'PooledResourceNotFound', 'CompositeResourceDefinitionNotFound',
      # 'CompositeResourceDefinitionNotValid', or 'CompositeResourceDefinitionNotRead
y'.
      # Not set if status: "True".
      reason:

  # (Optional) claimParameters contains the OpenAPIV3Schema used to configure
  # claims for this class.
  # Not set on pool-based classes.
  claimParameters:
    openAPIV3Schema:
      description: The OpenAPIV3Schema of this Composite Resource Definition.
      properties:
        storageGB:
          default: 1
          description: The desired storage capacity of the database, in Gigabytes.
          type: integer
      type: object

  # instanceType holds information about the resource selected by this class.
  # If using the Crossplane provisioner, this refers to the CompositeResource (XR)
  # defined by the CompositeResourceDefinition (XRD) referred to in the class.
  instanceType:
    # (Optional) Group specifies the API group.
    group: bitnami.database.tanzu.vmware.com
    # Kind specifies the API Kind.
    kind: XMySQLInstance
    # Version specifies the API version.
    version: v1alpha1

  # Populated based on metadata.generation when controller observes a change to
  # the resource. If this value is out of date, other status fields do not
  # reflect latest state.
  observedGeneration: 1
```

ClassClaim

`ClassClaim` refers to a `ClusterInstanceClass` from which service instances are then either selected (for pool-based classes) or provisioned (for provisioner-based classes) to fulfill the claim. `ClassClaim` adheres to Provisioned Service as defined by the Service Binding Specification for Kubernetes.You can bind a `ClassClaim` to an application workload by using a reference in the workload's `.spec.serviceClaims` configuration.

The following snippet outlines the `ClassClaims` YAML:

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ClassClaim

metadata:
  # The name for the claim.
  name: mysql-claim-1
  # The namespace in which to create the claim.
  namespace: my-apps

spec:
  # classRef holds a reference to a ClusterInstanceClass.
  classRef:
    # The name of the class from which to claim a service instance.
    # For information about the permissions users must have to claim from the class,
    # see the note below this snippet.
    name: mysql-unmanaged

  # (Optional) parameters are key-value pairs that are configuration inputs to the
  # instance obtained from the referenced ClusterInstanceClass. These parameters
  # only take effect when referring to a provisioner-based class.
  parameters:
    key: value

status:
  # Conditions for the claim.
  conditions:
    # The condition type. Can be one of 'Ready', 'ClassMatched', 'Validated', or
    # 'ResourceClaimCreated'. All condition types are initialized for all claims.
    # The Ready condition reports status: "True" once all other condition types are he
althy.
    - type: Ready
      # status can be either 'True' or 'False'.
      status: "True"
      # reason provides a reason for status: "False" for additional context.
      # One of 'UnableToFetchClass', 'ClassDoesNotExist', 'ClassNotReady',
      # 'UnableToQueryClaimableInstances', 'NoClaimableInstances',
      # 'UnableToCreateResourceClaim', 'UnableToCreateClaimableInstance', 'ResourceRea
dy',
      # 'ResourceNotReady', 'ResourceReadyUnsupported', or 'ReasonParametersInvalid'.
      # Not set if status: "True".
      reason:

  # binding holds a reference to a secret, in the same namespace, which contains
  # credentials for accessing the claimed service instance.
  binding:
    # The name of the `Secret`. The presence of the .status.binding.name field
    # marks this resource as a Provisioned Service.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c

  # provisionedResourceRef contains a reference to the provisioned resource.
  # Only set if the claim refers to a provisioner-based class.
  provisionedResourceRef:
    # The API Group/Version of the provisioned resource in the GROUP/VERSION format.
    apiVersion: bitnami.database.tanzu.vmware.com/v1alpha1
    # The API kind of the provisioned resource.
    kind: XMysqlInstance
    # The name of the provisioned resource.
```

```
    name: mysql-1-57dr7

  # resourceRef contains a reference to the claimed resource.
  resourceRef:
    # The API Group/Version of the claimed resource in the GROUP/VERSION format.
    apiVersion: v1
    # The API kind of the claimed resource.
    kind: Secret
    # The name of the claimed resource.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c
    # The namespace of the claimed resource.
    namespace: my-apps

  # Populated based on metadata.generation when controller observes a change to
  # the resource. If this value is out of date, other status fields do not reflect
  # latest state.
  observedGeneration: 1
```

Note

If you refer to a provisioner-based class in `spec.classref.name`, you must have sufficient RBAC permission to claim from the class. For more information, see Authorize users and groups to claim from provisioner-based classes.

ResourceClaim and ResourceClaimPolicy

This topic provides Services Toolkit API documentation for `ResourceClaim` and `ResourceClaimPolicy`.

ResourceClaim

`ResourceClaim` is used to claim a specific Kubernetes resource by using a reference. `ResourceClaim` adheres to Provisioned Service as defined by the Service Binding Specification for Kubernetes. you can bind a `ResourceClaim` to an application workload by using a reference in the workload's `.spec.serviceClaims` configuration.

A `ResourceClaim` is exclusive by nature. This means that after a `ResourceClaim` has claimed a resource, no other `ResourceClaim` can claim that same resource.

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaim

metadata:
  # The name for the claim.
  name: claim-1
  # The namespace in which to create the claim.
  namespace: my-apps
  # Internal finalizers applied by the resource claim controller to ensure
  # resources are cleaned up.
  finalizers:
  - resourceclaims.services.apps.tanzu.vmware.com/finalizer
  - resourceclaims.services.apps.tanzu.vmware.com/lease-finalizer

spec:
  # ref is a reference to the resource to be claimed.
  ref:
    # The API Group/Version of the resource to claim in the GROUP/VERSION format.
    apiVersion: v1
    # The API Kind of the resource to claim.
    kind: Secret
```

```
    # The name of the resource to claim.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c
    # (Optional) The namespace of the resource to claim. If the resource exists
    # in a different namespace to the namespace of the claim, then you must configure
    # a corresponding ResourceClaimPolicy to permit claiming of the resource.
    namespace: service-instances

status:
  # Conditions for the claim.
  conditions:
    # The condition type. Can be one of 'Ready', 'ResourceMatched' or 'ResourceMatche
d'.
    # All condition types are initialized for all claims.
    # The Ready condition reports status: "True" once all other condition types are he
althy.
    - type: Ready
      # status can be either 'True' or 'False'.
      status: "True"
      # reason provides a reason for status: "False" for additional context.
      # One of 'ResourceNotFound', 'BindingNotCopyable', 'UnableToSetExclusiveClaim',
      # 'ResourceNonBindable', 'NoMatchingResourceClaimPolicy',
      # 'UnableToTrackReferencedResource', 'ResourceAlreadyClaimed',
      # 'UpdatedResourceReference', or 'ClaimMarkedForDeletion'.
      # Not set if status: "True".
      reason:

  # binding holds a reference to a secret in the same namespace which contains
  # credentials for accessing the claimed service instance.
  binding:
    # The name of the secret. The presence of the .status.binding.name field marks
    # this resource as a Provisioned Service.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c

  # claimedResourceRef holds a reference to the claimed resource.
  claimedResourceRef:
    # The API Group/Version of the claimed resource in the GROUP/VERSION format.
    apiVersion: v1
    # The API kind of the claimed resource.
    kind: Secret
    # The name of the claimed resource.
    name: 770845b6-02f0-4c1b-8d0c-3dae81bad35c
    # The namespace of the claimed resource.
    namespace: service-instances

  # Populated based on metadata.generation when controller observes a change to
  # the resource. If this value is out of date, other status fields do not
  # reflect latest state.
  observedGeneration: 1
```

ResourceClaimPolicy

`ResourceClaimPolicy` provides a mechanism to either permit or deny the claiming of resources across namespaces.

```
apiVersion: services.apps.tanzu.vmware.com/v1alpha1
kind: ResourceClaimPolicy

metadata:
  # The name for the policy.
  name: default-ns-can-claim-secret-1
  # The namespace for the policy.
  # ResourceClaimPolicy resources must exist in the same namespace as the resources
  # they are permitting to be claimed.
  namespace: x-namespace-1
```

```
spec:
  # consumingNamespaces specifies the source namespace(s) to permit the claiming
  # of the resources from.
  # Use '*' to configure all namespaces.
  consumingNamespaces:
  - default

  # The API group of the resource to permit the claiming of.
  group: rabbitmq.com
  # The API kind of the resource to permit the claiming of.
  kind: RabbitmqCluster
  # (Optional) selector is a labelSelector to match resources to permit the claiming o
f.
  selector:
    matchLabels:
      "key": "value"
```

InstanceQuery

This topic provides Services Toolkit API documentation for `InstanceQuery`.

InstanceQuery

`InstanceQuery` is a create-only API that, given a pool-based `ClusterInstanceClass`, returns the intersection of the set of service instances represented by that class and the claimable service instances for the namespace of the `InstanceQuery`.

```
apiVersion: claimable.services.apps.tanzu.vmware.com/v1alpha1
kind: InstanceQuery

metadata:
  # An arbitrary name for the query.
  name: test
  # The namespace from which to run the query. The resulting list of instances is
  # specific to the namespace of the query itself.
  namespace: my-apps

spec:
  # The name of the class to query for claimable instances. Must refer to a
  # pool-based class and not a provisioner-based class.
  class: pooled-class-1
  # (Optional) A limit on the maximum number of instances to return.
  # The default is 50.
  limit: 1

status:
  # A list of service instances that you can claim by using ResourceClaims created
  # in the same namespace as the query.
  instances:
    # The API group/version of the claimable instance in the format GROUP/VERSION.
  - apiVersion: v1
    # The API kind of the claimable instance.
    kind: Secret
    # The name of the claimable instance.
    name: my-secret-two
    # The namespace of the claimable instance.
    namespace: default
```

RBAC

This topic provides API documentation for Role-Based Access Control (RBAC) relating to Services Toolkit's APIs.

Aggregation labels

This section describes the following Aggregation labels:

- servicebinding.io/controller: "true"
- services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"
- services.tanzu.vmware.com/aggregate-to-provider-helm: "true"

servicebinding.io/controller: "true"

Use this label to grant the Services Toolkit and service bindings controllers permission to get, list, and watch resources to be claimed and bound in the cluster.

For example, the following `ClusterRole` grants the controllers permission to get, list, and watch `RabbitmqCluster` resources. You cannot create `ClassClaims` or `ResourceClaims` unless the controllers have at least these permissions for each resource type being claimed.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: resource-claims-rmq-role
  labels:
    servicebinding.io/controller: "true"
rules:
- apiGroups:
  - rabbitmq.com
  resources:
  - rabbitmqclusters
  verbs:
  - get
  - list
  - watch
```

services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"

Use this label to aggregate RBAC rules to `provider-kubernetes`, which is a Crossplane `Provider` installed by default as part of the Crossplane package in Tanzu Application Platform. You must grant relevant RBAC permissions for each API Group/Kind used during the creation of `Compositions` as part of setting up dynamic provisioning.

For example, the following `ClusterRole` grants `provider-kubernetes` full control over `rabbitmqclusters` on the `rabbitmq.com` API Group. This allows you to compose `rabbitmqclusters` in `Compositions`. For a full example, see Setup Dynamic Provisioning of Service Instances.

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rmqcluster-read-writer
  labels:
    services.tanzu.vmware.com/aggregate-to-provider-kubernetes: "true"
rules:
- apiGroups:
  - rabbitmq.com
  resources:
  - rabbitmqclusters
```

```
  verbs:
  - "*"
```

services.tanzu.vmware.com/aggregate-to-provider-helm: “true”

Use this label to aggregate RBAC rules to `provider-helm`, which is a Crossplane `Provider` installed by default as part of the Crossplane package in Tanzu Application Platform. You must grant relevant RBAC permissions for each API Group/Kind used during the creation of Helm releases when using the `Release` managed resource as part of `Compositions`.

For example, the following `ClusterRole` grants `provider-helm` full control over `rabbitmqclusters` on the `rabbitmq.com` API Group. This allows you to compose Helm `Releases` which themselves eventually deploy `rabbitmqclusters` in your `Compositions`.

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: rmqcluster-read-writer
  labels:
    services.tanzu.vmware.com/aggregate-to-provider-helm: "true"
rules:
- apiGroups:
  - rabbitmq.com
  resources:
  - rabbitmqclusters
  verbs:
  - "*"
```

The claim verb for ClusterInstanceClass

Services Toolkit supports using the `claim` verb for RBAC rules that apply to a `ClusterInstanceClass`. You can use this with relevant aggregating labels or `ClusterRoleBindings` as a form of access control to specify who can claim from which `ClusterInstanceClass`.

For example:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: app-operator-claim-class-bigcorp-rabbitmq
  labels:
    # (Optional) Aggregates this ClusterRole to Tanzu Application Platform's
    # app-operator user role at the cluster scope. You can choose to aggregate
    # this to any of the other standard user roles as well.
    apps.tanzu.vmware.com/aggregate-to-app-operator-cluster-access: "true"
rules:
# Permits claiming from the 'bigcorp-rabbitmq' class
- apiGroups:
  - services.apps.tanzu.vmware.com
  resources:
  - clusterinstanceclasses
  resourceNames:
  - bigcorp-rabbitmq
  verbs:
  - claim
```

As of Services Toolkit v0.10.3 in Tanzu Application Platform v1.5.6, you must grant the permission to `claim` from a `ClusterInstanceClass` at the cluster level. You now must use a `ClusterRole` and `ClusterRoleBinding`. Namespace-scoped permissions, such as using a `Role` and `RoleBinding` or

`ClusterRole` and `RoleBinding`, are not sufficient. If you used `Roles` and `RoleBindings`, or `ClusterRoles` and `RoleBindings` to grant `claim` permissions in specific namespaces only, this change might affect you. For more information, see Authorize users and groups to claim from provisioner-based classes.

Previously, Services Toolkit allowed you to `claim` from a `ClusterInstanceClass` with only namespace-level permissions. However, this allowed users with only namespace-level permissions to obtain or indirectly deploy resources into namespaces that they do not have access to according to the RBAC permissions.

Services Toolkit limitations

This topic tells you about the limitations related to working with services on Tanzu Application Platform (commonly known as TAP).

Cannot claim and bind to the same service instance from across multiple namespaces

Two or more workloads located in two or more distinct namespaces cannot claim and bind to the same service instance. This is due to the mutually exclusive nature of claims. After a claim has claimed a service instance, no other claim can then claim that same service instance.

This limitation does not exist for two or more workloads located in the same namespace. In this case, the workloads can all still all bind to one claim. This is not possible across the namespace divide.

Tanzu Service CLI plug-in

This topic provides reference information about the Tanzu Service CLI plug-in for Services Toolkit. The main use for the plug-in is for application operators and application developers to create claims. It aims to offer you a service experience that is consistent with other Tanzu CLI commands.

The reference material in this topic is split by sub-command.

tanzu service class

Classes, sometimes called instance classes or service instance classes, are a means to discover and describe groupings of similar service instances. They are analogous to the concept of storage classes in Kubernetes.

You can discover the range of services on offer by listing the available classes on a cluster. See `tanzu service class list -h`.

You can create a claim for a service instance of a particular class by running the `tanzu service class-claim create` command.

When you get a class, you can see more detailed information about it, including, where available, a list of parameters that you can pass to the `tanzu service class-claim create` command using the `--parameter` flag.

tanzu service class list

This command lists the available classes.

```
Usage:
  tanzu service class list [flags]
```

```
Examples:
  tanzu service class list

Flags:
  -h, --help   help for list

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service class get

This command gets detailed information for a class.

The output includes more detailed information about the class, including, where available, a list of parameters that you can pass to the `tanzu service class-claim create` command using the `--parameter` flag.

```
Usage:
  tanzu service class get [name] [flags]

Examples:
  tanzu service class get rmq-small

Flags:
  -h, --help   help for get

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service class-claim

Class claims allow you to create claims by only referring to a class.

Class claims are an alternative approach to resource claims, which require you to refer to a specific resource by name, namespace, kind and API group/version.

VMware recommends that you work with class claims wherever possible because they are easier to create and are considered more portable across multiple clusters.

tanzu service class-claim create

This command creates a claim by referring to a class.

You can bind claims for service instances to application workloads.

Claims are mutually exclusive, meaning that after a service instance has been claimed, no other claim can claim it. This prevents unauthorized application workloads from accessing a service instance that your application workloads are using.

You can pass parameters with the `--parameter key.subKey=value` flag. You can provide this flag multiple times. The value must be valid YAML. You can find available parameters for a class by running `tanzu service class get CLASS-NAME`.

```
Usage:
  tanzu service class-claim create [name] [flags]
```

```
Examples:
  tanzu service class-claim create psql-claim-1 --class postgres
  tanzu service class-claim create rmq-claim-1 --class rmq --parameter durable=true --
parameter replicas=3

Flags:
      --class string            the name of a class to claim an instance of
  -h, --help                    help for create
  -n, --namespace name          kubernetes namespace (defaulted from kube config)
  -p, --parameter stringArray   claim parameters

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service class-claim get

This command gets detailed information for a class claim.

The output includes the name of the class the claim was created for and the claim ref. Pass claim refs to the `--service-ref` flag of the `tanzu apps workload create` command to bind workloads to claimed service instances.

```
Usage:
  tanzu service class-claim get [flags]

Examples:
  tanzu service class-claim get psql-claim-1
  tanzu service class-claim get psql-claim-1 --namespace app-ns-1

Flags:
  -h, --help             help for get
  -n, --namespace name   kubernetes namespace (defaulted from kube config)

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service class-claim delete

This command deletes a class claim.

You will be prompted to confirm the deletion unless you pass the `--yes` flag. Before you delete a claim, you must be aware of the consequences of doing so.

When you create a claim, it signals a that you want a service instance. You usually create a service instance to bind it to one or more application workload. If you delete a claim, it signals that you no longer need the claimed service instance. At this point, other claims created by other users can claim the service instance you previously claimed.

```
Usage:
  tanzu service class-claim delete [flags]

Examples:
  tanzu service class-claim delete psql-claim-1
  tanzu service class-claim delete psql-claim-1 --yes
  tanzu service class-claim delete psql-claim-1 --namespace app-ns-1

Flags:
```

```
  -h, --help              help for delete
  -n, --namespace name    kubernetes namespace (defaulted from kube config)
  -y, --yes               skip the confirmation of the deletion

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service class-claim list

This command lists class claims in a namespace or across all namespaces.

If you run this command with the `-o wide` flag, claim refs for each of the claims are printed. Pass claim refs to the `--service-ref` flag of the `tanzu apps workload create` command to bind workloads to claimed service instances.

```
Usage:
  tanzu service class-claim list [flags]

Examples:
  tanzu service class-claim list
  tanzu service class-claim list --class postgres
  tanzu service class-claim list -o wide
  tanzu service class-claim list -n app-ns-1 -o wide

Flags:
  -A, --all-namespaces    list class claims across all namespaces
  -c, --class string      list class claims referencing this class
  -h, --help              help for list
  -n, --namespace name    kubernetes namespace (defaulted from kube config)
  -o, --output string     output format (currently the only available option is 'wide')

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service resource-claim

Resource claims enable you to create claims by referring to a specific resource by name, namespace, kind, and API group or version.

Resource claims are an alternative approach to class claims, which only require you to refer to a class.

VMware recommends that you work with class claims wherever possible because they are easier to create and are more portable across multiple clusters.

tanzu service resource-claim create

This command creates a claim for a specific resource.

It is common to create claims for resources that you can bind to application workloads using the claim.

This approach to creating claims differs to that of class claims, in which the system ultimately finds and supplies a claimable resource for you. You only have to work with resource claims if you want full control over which resource is claimed. If not, it is simpler and more convenient to work with class claims. See `tanzu service class-claim --help`.

Claims are mutually exclusive, meaning that after a service instance has been claimed, no other claim can claim it. This prevents unauthorized application workloads from accessing a resource that your application workloads are using.

To find resources you can create resource claims for, run the `tanzu service claimable list` command.

```
Usage:
  tanzu service resource-claim create [name] [flags]

Examples:
  tanzu service resource-claim create psql-claim-1 --resource-name psql-instance-1 --r
esource-kind Postgres --resource-api-version sql.example.com/v1
  tanzu service resource-claim create psql-claim-1 --resource-name psql-instance-1 --r
esource-kind Postgres --resource-api-version sql.example.com/v1 --resource-namespace s
ervice-instances-1
  tanzu service resource-claim create psql-claim-1 --resource-name secret-1 --resource
-kind Secret --resource-api-version v1

Flags:
  -h, --help                          help for create
  -n, --namespace name                kubernetes namespace (defaulted from kube confi
g)
      --resource-api-version string   API group and version of the resource to claim
(in the form '<GROUP>/<VERSION>')
      --resource-kind string          kind of the resource to claim
      --resource-name string          name of the resource to claim
      --resource-namespace string     namespace of the resource to claim

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service resource-claim get

This command gets detailed information for a resource claim.

The output includes the name of claimed resource and the claim ref. Pass claim refs to the `--service-ref` flag of the `tanzu apps workload create` command to bind workloads to claimed service instances.

```
Usage:
  tanzu service resource-claim get [name] [flags]

Examples:
  tanzu service resource-claim get psql-claim-1
  tanzu service resource-claim get psql-claim-1 --namespace app-ns-1

Flags:
  -h, --help             help for get
  -n, --namespace name   kubernetes namespace (defaulted from kube config)

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service resource-claim delete

This command deletes a resource claim.

You will be prompted to confirm the deletion unless you pass the `--yes` flag. Before you delete a claim, you must be aware of the consequences of doing so.

When you create a claim, it signals a that you want a resource. You usually create a resource to bind it to one or more application workload. If you delete a claim, it signals that you no longer need the claimed resource. At this point, other claims created by other users can claim the resource you previously claimed.

```
Usage:
  tanzu service resource-claim delete [name] [flags]

Examples:
  tanzu service resource-claim delete psql-claim-1
  tanzu service resource-claim delete psql-claim-1 --yes
  tanzu service resource-claim delete psql-claim-1 --namespace app-ns-1

Flags:
  -h, --help              help for delete
  -n, --namespace name    kubernetes namespace (defaulted from kube config)
  -y, --yes               skip the confirmation of the deletion

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service resource-claim list

This command lists resource claims in a namespace or across all namespaces.

If you run this command with the `-o wide` flag, claim refs for each of the claims are printed. Pass claim refs to the `--service-ref` flag of the `tanzu apps workload create` command to bind workloads to claimed service instances.

```
Usage:
  tanzu service resource-claim list [flags]

Examples:
  tanzu service resource-claim list
  tanzu service resource-claim list -o wide
  tanzu service resource-claim list -n app-ns-1 -o wide

Flags:
  -A, --all-namespaces   list resource claims across all namespaces
  -h, --help             help for list
  -n, --namespace name   kubernetes namespace (defaulted from kube config)
  -o, --output string    output format (currently the only available option is 'wide')

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

tanzu service claimable

Searches for resources that are available to claim.

tanzu service claimable list

This command lists resources for a class that you can claim directly using the `tanzu service resource-claim create` command.

```
Usage:
  tanzu service claimable list [flags]

Examples:
  tanzu service claimable list --class postgres
  tanzu service claimable list --class postgres --namespace app-ns-1

Flags:
      --class string     name of the class to list claimable resources for
  -h, --help             help for list
  -n, --namespace name   kubernetes namespace (defaulted from kube config)

Global Flags:
      --context string      name of the kubeconfig context to use (default is current-
context defined by kubeconfig)
      --kubeconfig string   kubeconfig file (default is /home/eking/.kube/config)
      --no-color            turn off color output in terminals
```

Services Toolkit terminology and user roles

This topic provides descriptions of the terms and user roles used in the Services Toolkit documentation.

Terminology

The following terms are used in the Services Toolkit documentation.

Service

Service is broad, high-level term that describes something used in either the development of, or running of application workloads. Often, but not exclusively, synonymous with the concept of a backing service as defined by the Twelve Factor App:

“… any service the app consumes over the network as part of its normal operation”

For example:

- A PostgreSQL service (implemented as a Kubernetes Operator provided by Tanzu Data Services)
- A PostgreSQL service (implemented as a process running on an Application Developer’s laptop)
- Object storage (implemented as SaaS running on AWS)
- AppSSO

Service resource

A service resource is a Kubernetes resource that provides some of the functions related to a Service.

For example:

- A Kubernetes resource with API Kind `PostgreSQL`
- A Kubernetes resource with API Kind `FirewallRule`
- A Kubernetes resource with API Kind `RabbitmqUser`

- A Kubernetes resource with API Kind `ClientRegistration` that provides access to an App SSO service
- A Kubernetes resource with API Kind `Secret` containing credentials and connectivity information for a Service that may or may not be running on the cluster itself.

Provisioned service

A provisioned service is any service resource that defines a `.status.binding.name` which points to a secret in the same namespace that contains credentials and connectivity information for the resource.

This term is defined in the Service Binding Specification for Kubernetes. For the full definition, see the Service Binding Specification in GitHub.

Service binding

A service binding is a mechanism in which service instance credentials and other related connectivity information are automatically communicated to application workloads.

For example:

- The Service binding concept implemented through the `ServiceBinding` service resource provided by servicebinding in GitHub.

Service instance

A service instance is an abstraction over one or a group of interrelated service resources that together provide the functions for a particular service.

One of the service resources that make up an instance must either adhere to the definition of provisioned service, or be a secret conforming to the service binding specification for Kubernetes. This guarantees that you can claim a service and subsequently bind service instances to application workloads.

You make service instances discoverable through service instance classes.

For example:

- The `RabbitmqCluster` service resource provided by the RabbitMQ Cluster Kubernetes operator. This service resource adheres to provisioned service. Therefore, you can consider any `RabbitmqCluster` resource on a Kubernetes cluster to be a service instance.
- A logical grouping of the following service resources form a single AWS RDS service instance:
    - An AWS RDS `DBInstance`
    - An AWS RDS `DBSubnetGroup`
    - A Carvel `SecretTemplate` configured to produce a secret conforming to the Service Binding Specification for Kubernetes
    - A `Role`, `RoleBinding`, and `ServiceAccount`
- A Kubernetes `Secret` conforming to the Service Binding Specification for Kubernetes containing credentials for a Service running external to the cluster.

Service instance class

A service instance class is more commonly called a “class”. They provide a way to describe classes, that is, categories, of service instances.

A service instance class enables service instances belonging to the class to be discovered. They come in one of two varieties: pool-based or provisioner-based.

- Claims for pool-based classes are fulfilled by selecting a service instance from a pool.
- Claims for provisioner-based classes are fulfilled by provisioning new service instances.

Different classes might map to different services or to different configurations of the same service.

For example:

- A `ClusterInstanceClass` named “rabbitmq-dev” pointing to all `RabbitmqCluster` service resources configured with `.spec.replicas=1` identified by label `class: rmq-dev`.
- A `ClusterInstanceClass` named “rabbitmq-prod” pointing to all `RabbitmqCluster` service resources configured with `.spec.replicas=3` identified by label `class: rmq-prod`.
- A `ClusterInstanceClass` named “aws-rds-postgresql” pointing to secrets that conform with the Binding Specification and identified by label `class: aws-rds`.
- A `ClusterInstanceClass` named “mysql-on-demand” which provisions MySQL service instances.

Claim

A claim is a mechanism in which requests for service instances can be declared and fulfilled without requiring detailed knowledge of the service instances themselves.

Claims come in one of two varieties - resource claim and class claim:

- Resource claims refer to a specific service instance.
- Class claims refer to a class from which a service instance is then either selected (pool-based) or provisioned (provisioner-based).

For example:

- A resource claim pointing to a `RabbitmqCluster` service instance named `rmq-1` in the namespace `service-instances`.
- A class claim pointing to a class named `on-demand-rabbitmq`.

Claimable service instance

A claimable service instance is any service instance that you are permitted to claim using a resource claim from a namespace, taking into consideration:

- Location (namespace) of the service instance in relation to the location of the resource claim.
- Any matching resource claim policies.
- Exclusivity of resource claims, that is, you can only claim an instance once.

For example:

- A `RabbitmqCluster` service resource located in the same namespace as a resource claim and that has not already been claimed by another resource claim is a claimable service instance.
- A `RabbitmqCluster` service resource located in a different namespace to a resource claim, for which a matching resource claim policy exists, and has not already been claimed by another resource claim is a claimable service instance.
- A `RabbitmqCluster` service resource located in the same namespace as a resource claim that has already been claimed is not a claimable service instance due to the exclusive

nature of Resource Claims.

Dynamic provisioning

Dynamic provisioning is a capability of Services Toolkit in which class claims that refer to provisioner-based classes are fulfilled automatically through the provisioning of new service instances.

Service resource life cycle API

A service resource life cycle API is any Kubernetes API that you can use to manage the life cycle—create, read, update and delete (CRUD)—of a service resource.

For example:

- `rabbitmqclusters.rabbitmq.com/v1beta1`

Service cluster

A service cluster is applicable within the context of Service API Projection and Service Resource Replication. It is a Kubernetes cluster that has Service Resource Lifecycle APIs installed and a corresponding controller managing their life cycle.

Workload cluster

A workload cluster is applicable within the context of Service API Projection and Service Resource Replication. It is a Kubernetes cluster that has developer-created applications running on it.

User roles

Services Toolkit caters to the following user roles.

These user roles are not user personas. It is possible, and even expected, that one person can be associated with many user roles at any given time. The user roles align to Tanzu Application Platform's user roles. Services Toolkit is primarily responsible for defining the service operator role. For more information about the user roles, see Role descriptions.

The user roles listed in this section consist of a short description and the tasks required. For detailed information about the corresponding Role-Based Access Control (RBAC) associated with each role, see Detailed role permissions breakdown.

Application developer (AD)

The application developer role encompasses both app-editor and app-viewer roles as defined by Tanzu Application Platform. For more information about the app-editor and app-viewer roles, see Role descriptions.

Application developers do the following:

- Bind and unbind application workloads to and from resource claims.
- Get, list, and watch ResourceClaims.
- get, list, and watch ClusterInstanceClasses associated with ResourceClaims.

Application operator (AO)

Encompasses the app-operator role as defined by Tanzu Application Platform. For more information about the app-operator role, see Role descriptions.

Application operators do the following:

- Discover and learn about service instance classes available on a cluster.
- Discover claimable service instances associated with service instance classes.
- Life cycle management (CRUD) of resource claims.

Service operator (SO)

Service operators do the following:

- Life cycle management (CRUD) of service instances.
- Life cycle management (CRUD) of service instance classes.
- Life cycle management (CRUD) of resource claim policies.
- Identify pending resource claims and, if appropriate, help to fulfil such claims through a combination of the previous tasks.
- Setup and configure dynamic provisioning.

Overview of Source Controller

Tanzu Source Controller provides a standard interface for artifact acquisition and extends the function of Flux CD Source Controller.

Tanzu Source Controller supports the following two resource types:

- ImageRepository
- MavenArtifact

An `ImageRepository` resource can resolve the source from the contents of an image in an image registry. This enables app developers to create and update workloads from local source code or a code repository.

A `MavenArtifact` resource can resolve a binary artifact from a Maven repository. This functionality enables the supply chain to support artifacts produced externally.

Note

Fetching `RELEASE` version from GitHub packages is not currently supported. The `metadata.xml` in GitHub packages does not have the `release` tag that contains the released version number. For more information, see Maven-metadata.xml is corrupted on upload to registry on GitHub.

Overview of Source Controller

Tanzu Source Controller provides a standard interface for artifact acquisition and extends the function of Flux CD Source Controller.

Tanzu Source Controller supports the following two resource types:

- ImageRepository
- MavenArtifact

An `ImageRepository` resource can resolve the source from the contents of an image in an image registry. This enables app developers to create and update workloads from local source code or a code repository.

A `MavenArtifact` resource can resolve a binary artifact from a Maven repository. This functionality enables the supply chain to support artifacts produced externally.

Note

Fetching `RELEASE` version from GitHub packages is not currently supported. The `metadata.xml` in GitHub packages does not have the `release` tag that contains the released version number. For more information, see Maven-metadata.xml is corrupted on upload to registry on GitHub.

Install Source Controller

This topic tells you how to install Source Controller from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Source Controller. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Source Controller:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager on the cluster. For more information, see Install cert-manager.

Install

To install Source Controller:

1. List version information for the package by running:

```
tanzu package available list controller.source.apps.tanzu.vmware.com --namespac
e tap-install
```

For example:

```
$ tanzu package available list controller.source.apps.tanzu.vmware.com --namesp
ace tap-install
- Retrieving package versions for controller.source.apps.tanzu.vmware.com...
  NAME                                     VERSION  RELEASED-AT
  controller.source.apps.tanzu.vmware.com  0.3.1    2022-01-23 19:00:00 -0500 -
05
  controller.source.apps.tanzu.vmware.com  0.3.2    2022-02-21 19:00:00 -0500 -
05
  controller.source.apps.tanzu.vmware.com  0.3.3    2022-03-03 19:00:00 -0500 -
05
  controller.source.apps.tanzu.vmware.com  0.4.1    2022-06-09 19:00:00 -0500 -
05
```

2. (Optional) Gather the values schema:

```
tanzu package available get controller.source.apps.tanzu.vmware.com/VERSION-NUM
BER --values-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed in step 1 above.

For example:

```
tanzu package available get controller.source.apps.tanzu.vmware.com/0.4.1 --val
ues-schema --namespace tap-install

Retrieving package details for controller.source.apps.tanzu.vmware.com/0.4.1...
KEY               DEFAULT  TYPE    DESCRIPTION
aws_iam_role_arn           string  Optional: The AWS IAM Role ARN to attach to
the Source Controller service account
ca_cert_data               string  Optional: PEM Encoded certificate data for i
mage registries with private CA.
```

3. (Optional) Create a file named `source-controller-values.yaml` to override the default installation settings. You can configure the following fields:

    - `ca_cert_data`: Enables Source Controller to connect to image registries that use self-signed or private certificate authorities. If a certificate error `x509: certificate signed by unknown authority` occurs, use this option to trust additional certificate authorities.

      To provide a custom certificate, add the PEM-encoded CA certificate data to `source-controller-values.yaml`. For example:

      ```
      ca_cert_data: |
          -----BEGIN CERTIFICATE-----
          MIICpTCCAYUCBgkqhkiG9w0BBQ0wMzAbBgkqhkiG9w0BBQwwDgQIYg9x6gkCAggA
          ...
          9TlA7A4FFpQqbhAuAVH6KQ8WMZIrVxJSQ03c9lKVkI62wQ==
          -----END CERTIFICATE-----
      ```

    - `aws_iam_role_arn`: Annotates the Source Controller service with an AWS Identity and Access Management (IAM) role. This allows Source Controller to pull images from Amazon Elastic Container Registry (ECR).

      To add the AWS IAM role Amazon Resource Name (ARN) to the Source Controller service, add the ARN to `source-controller-values.yaml`. For example:

      ```
      aws_iam_role_arn: "eks.amazonaws.com/role-arn: arn:aws:iam::112233445566:
      role/source-controller-manager"
      ```

4. Install the package by running:

    ```
    tanzu package install source-controller \
      --package controller.source.apps.tanzu.vmware.com \
      --version VERSION-NUMBER \
      --namespace tap-install \
      --values-file VALUES-FILE
    ```

    Where:

    - `VERSION-NUMBER` is the version of the package listed in step 1 above.
    - `VALUES-FILE` is the path to the file created in step 3.

    For example:

    ```
    $ tanzu package install source-controller
        --package controller.source.apps.tanzu.vmware.com
        --version 0.4.1
        --namespace tap-install
        --values-file source-controller-values.yaml
    ```

```
\ Installing package 'controller.source.apps.tanzu.vmware.com'
| Getting package metadata for 'controller.source.apps.tanzu.vmware.com'
| Creating service account 'source-controller-default-sa'
| Creating cluster admin role 'source-controller-default-cluster-role'
| Creating cluster role binding 'source-controller-default-cluster-rolebinding'
| Creating secret 'source-controller-default-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'source-controller'
- 'PackageInstall' resource install status: Reconciling

 Added installed package 'source-controller'
```

5. Verify the package installation by running:

```
tanzu package installed get source-controller -n tap-install
```

For example:

```
tanzu package installed get source-controller -n tap-install
- Retrieving installation details for source-controller...
NAME:                    source-controller
PACKAGE-NAME:            controller.source.apps.tanzu.vmware.com
PACKAGE-VERSION:         0.4.1
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`:

```
kubectl get pods -n source-system
```

For example:

```
$ kubectl get pods -n source-system
NAME                                        READY   STATUS    RESTARTS   AGE
source-controller-manager-f68dc7bb6-4lrn6   1/1     Running   0          100s
```

Verify that `STATUS` is `Running`.

Troubleshoot Source Controller

This topic gives you guidance about how to troubleshoot issues with Source Controller.

Collecting Logs from Source Controller Manager

To retrieve Pod logs from the `controller-manager`, run the following command in the `source-system` namespace:

```
kubectl logs -n source-system -l control-plane=controller-manager
```

For example:

```
kubectl logs -n source-system -l control-plane=controller-manager
2021-11-18T17:59:43.152Z        INFO    controller.imagerepository      Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.metarepository       Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.metarepository       Starting Event
```

```
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.metarepository       Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.metarepository       Starting Contr
oller   {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository"}
2021-11-18T17:59:43.152Z        INFO    controller.imagerepository      Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.imagerepository      Starting Event
Source  {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "source": "kind source: /, Kind="}
2021-11-18T17:59:43.152Z        INFO    controller.imagerepository      Starting Contr
oller   {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository"}
2021-11-18T17:59:43.389Z        INFO    controller.metarepository       Starting worke
rs      {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "MetaR
epository", "worker count": 1}
2021-11-18T17:59:43.391Z        INFO    controller.imagerepository      Starting worke
rs      {"reconciler group": "source.apps.tanzu.vmware.com", "reconciler kind": "Image
Repository", "worker count": 1}
```

Source Controller reference

This topic provides reference documentation for Source Controller.

ImageRepository

```
---
apiVersion: source.apps.tanzu.vmware.com/v1alpha1
kind: ImageRepository
spec:
  image: registry.example/image/repository:tag
  # optional fields
  interval: 5m
  imagePullSecrets: []
  serviceAccountName: default
```

`ImageRepository` resolves source code defined in an Open Container Initiative (OCI) image repository, exposing the resulting source artifact at a URL defined by `.status.artifact.url`.

The interval determines how often to check tagged images for changes. Setting this value too high will result in delays in discovering new sources, while setting it too low may trigger a registry's rate limits.

Repository credentials can be defined as image pull secrets. You can reference them either directly from the resources at `.spec.imagePullSecrets` or attach them to a service account referenced at `.spec.serviceAccountName`. The default service account name `"default"` is used if not otherwise specified. The default credential helpers for the registry are also used, for example, pulling from Google Container Registry (GCR) on a Google Kubernetes Engine (GKE) cluster.

MavenArtifact

```
---
apiVersion: source.apps.tanzu.vmware.com/v1alpha1
kind: MavenArtifact
metadata:
  name: mavenartifact-sample
```

```
spec:
  artifact:
    groupId: org.springframework.boot
    artifactId: spring-boot
    version: "2.7.0"
  repository:
    url: https://repo1.maven.org/maven2
  interval: 5m0s
  timeout: 1m0s
```

`MavenArtifact` resolves artifact from a Maven repository, exposing the resulting artifact at a URL defined by `.status.artifact.url`.

The `interval` determines how often to check artifact for changes. Setting this value too high results in delays in discovering new sources, while setting it too low may trigger a repository's rate limits.

Repository credentials may be defined as secrets referenced from the resources at `.spec.repository.secretRef`. Secrets referenced by `spec.repository.secretRef` is parsed as follows:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: auth-secret
type: Opaque
data:
  username: <BASE64>
  password: <BASE64>
  caFile:   <BASE64>   // PEM Encoded certificate data for Custom CA
  certFile: <BASE64>   // PEM-encoded client certificate
  keyFile:  <BASE64>   // Private Key
```

Maven supports a broad set of `version` syntax. Source Controller supports a strict subset of Maven's version syntax in order to ensure compatibility and avoid user confusion. The subset of supported syntax may grow over time, but will never expand past the syntax defined directly by Maven. This behavior means that we can use `mvn` as a reference implementation for artifact resolution.

Version support implemented in the following order:

1. Pinned version - an exact match of a version in2 `maven-metadata.xml (versioning/versions/version)`.
2. `RELEASE` - metaversion defined in `maven-metadata.xml (versioning/release)`.
3. `*-SNAPSHOT` - the newest artifact for a snapshot version.
4. `LATEST` - metaversion defined in `maven-metadata.xml (versioning/latest)`.
5. Version ranges - https://maven.apache.org/enforcer/enforcer-rules/versionRanges.html. Support is planned for a future release.

Note

Pinned versions should be immutable, all other versions are dynamic and can change at any time. The `.spec.interval` defines how frequently to check for updated artifacts.

Overview of Spring Boot conventions

This topic tells you about the Spring Boot convention server.

The Spring Boot convention server is a bundle of small conventions applied to any Spring Boot application that is submitted to the supply chain in which the convention controller is configured.

Run the `docker inspect` command to make the Spring Boot convention server look inside the image. Example command:

```
$ docker inspect springio/petclinic
```

Example output:

```
[
    {
        "Id": "sha256:...",
        "RepoTags": [
            "springio/petclinic:latest"
        ],
        "RepoDigests": [
            "springio/petclinic@sha256:..."
        ],
        "Parent": "",
        "Container": "",
        ...
        "ContainerConfig": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            ...
            "Labels": null
        },
        "DockerVersion": "",
        "Author": "",
        "Config": {
...
]
```

The convention server searches inside the image for `Config -> Labels -> io.buildpacks.build.metadata` to find the `bom` file. It looks inside the `bom` file for metadata to evaluate whether the convention is to be applied.

For the list of conventions, see Conventions.

Overview of Spring Boot conventions

This topic tells you about the Spring Boot convention server.

The Spring Boot convention server is a bundle of small conventions applied to any Spring Boot application that is submitted to the supply chain in which the convention controller is configured.

Run the `docker inspect` command to make the Spring Boot convention server look inside the image. Example command:

```
$ docker inspect springio/petclinic
```

Example output:

```
[
    {
        "Id": "sha256:...",
        "RepoTags": [
            "springio/petclinic:latest"
        ],
```

```
        "RepoDigests": [
            "springio/petclinic@sha256:..."
        ],
        "Parent": "",
        "Container": "",
        ...
        "ContainerConfig": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            ...
            "Labels": null
        },
        "DockerVersion": "",
        "Author": "",
        "Config": {
...
]
```

The convention server searches inside the image for `Config -> Labels -> io.buildpacks.build.metadata` to find the `bom` file. It looks inside the `bom` file for metadata to evaluate whether the convention is to be applied.

For the list of conventions, see Conventions.

Install Spring Boot conventions

This topic tells you how to install Spring Boot conventions from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Spring Boot conventions. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Spring Boot conventions:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Supply Chain Choreographer.

Install Spring Boot conventions

To install Spring Boot conventions:

1. Get the exact name and version information for the Spring Boot conventions package to install by running:

   ```
   tanzu package available list spring-boot-conventions.tanzu.vmware.com --namespa
   ce tap-install
   ```

   For example:

   ```
   $ tanzu package available list spring-boot-conventions.tanzu.vmware.com --names
   pace tap-install
   ```

```
/ Retrieving package versions for spring-boot-conventions.tanzu.vmware.com...
NAME                                        VERSION           RELEASED-AT
...
spring-boot-conventions.tanzu.vmware.com    1.4.0             2022-12-08T00:00:0
0Z
...
```

2. (Optional) Change the default installation settings by running:

```
tanzu package available get spring-boot-conventions.tanzu.vmware.com/VERSION-NU
MBER \
--values-schema --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package listed. For example: `1.5.12`.

For example:

```
$ tanzu package available get spring-boot-conventions.tanzu.vmware.com/1.4.0 --
values-schema --namespace tap-install
  KEY                              DEFAULT             TYPE     DESCRIPTION
    autoConfigureActuators         false               boolean  Enable or disa
ble the automatic configuration of actuators on the TAP platform level
    kubernetes_distribution                            string   Kubernetes dis
tribution that this package is being installed on. Accepted
                                                                values:
['''',''openshift'']
    kubernetes_version                                 string   Optional: The
Kubernetes Version. Valid values are '1.24.*', or ''
```

3. Install the package by running:

```
tanzu package install spring-boot-conventions \
 --package spring-boot-conventions.tanzu.vmware.com \
 --version VERSION-NUMBER \
 --namespace tap-install
```

Where `VERSION-NUMBER` is the version of the package you listed earlier.

4. Verify you installed the package by running:

```
tanzu package installed get spring-boot-conventions --namespace tap-install
```

For example:

```
tanzu package installed get spring-boot-conventions -n tap-install
| Retrieving installation details for spring-boot-conventions...
NAME:                    spring-boot-conventions
PACKAGE-NAME:            spring-boot-conventions.tanzu.vmware.com
PACKAGE-VERSION:         1.5.12
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`

Configure and access Spring Boot actuators in Tanzu Application Platform

This topic tells you how the Spring Boot conventions in Tanzu Application Platform configure Spring Boot actuators automatically. With this feature, users can activate or deactivate the automatic configuration of actuators on Tanzu Application Platform and on individual workloads.

Workload-level configuration

Developers can add a label to their workloads to activate or deactivate the automatic configuration of actuators. By default, all existing and future accelerator projects are configured to activate automatic configuration on the workload level.

To activate or deactivate the automatic configuration of actuators at the workload level, follow these steps:

1. To activate automatic configuration of actuators, set the following label to `true` in your workload YAML:

   ```
   apps.tanzu.vmware.com/auto-configure-actuators: "true"
   ```

   If the preceding label is set to `true`, the Spring Boot actuator convention sets the following actuator configuration:

   - The `JAVA_TOOL_OPTIONS` property is set as `-Dmanagement.server.port="8081"`.
   - The `JAVA_TOOL_OPTIONS` property is set as `-Dmanagement.endpoints.web.base-path="/actuator"`.
   - Annotation on the PodIntent is set as `boot.spring.io/actuator: http://:8081/actuator`.

   In addition to these settings, Application Live View is activated with the following actuator configuration:

   - Label on the PodIntent is set as `tanzu.app.live.view.application.actuator: actuator`.
   - Label on the PodIntent is set as `tanzu.app.live.view.application.actuator.port: 8081`.

2. To deactivate automatic configuration of actuators, set the following label to `false` in your workload YAML:

   ```
   apps.tanzu.vmware.com/auto-configure-actuators: "false"
   ```

   If the preceding label is set to `false`, the Spring Boot actuator convention does not set any `JAVA_TOOL_OPTIONS` and does not set the annotation `boot.spring.io/actuator`.

   Application Live View is activated and configured with default values for Spring Boot web applications, assuming that some actuators are activated on the default port. On activating Application Live View, the following actuator settings are set:

   - The `JAVA_TOOL_OPTIONS` property is set as `-Dserver.port="8080"`.
   - Label on the PodIntent is set as `tanzu.app.live.view.application.actuator: actuator`.
   - Label on the PodIntent is set as `tanzu.app.live.view.application.actuator.port: 8080`.

The Application Live View GUI renders the pages with accessible information based on whether the actuator endpoints are accessible for an application.

By default, as an additional security measure, Spring Boot conventions does not expose all the actuator data over HTTP by exposing all the actuator endpoints. In addition, the information exposed by the health endpoint is not set to `always` by default.

If the automatic configuration of actuators is set to `true` either at the workload level or platform level, the Spring Boot convention then sets the runtime environment properties `management.endpoints.web.exposure.include="*"` and `management.endpoint.health.show-details=true` on to the PodSpec to expose all the actuator endpoints and detailed health

information. You do not need to add these properties manually in `application.properties` or `application.yml`.

Platform-level configuration

In contrast to activating or deactivating the automatic configuration of actuators on the level of individual workloads, you can set a global setting for the platform instead. This setting is taken into account ONLY when there is no specific `auto-configure-actuators` setting on the individual workload.

To activate or deactivate the automatic configuration of actuators at a global level, follow these steps:

1. When you install Spring Boot conventions, you can provide an entry in the `values.yaml` file to activate automatic configuration. For example:

   ```
   springboot_conventions:
     autoConfigureActuators: true
   ```

2. To deactivate the automatic configuration, you can provide the following entry:

   ```
   springboot_conventions:
     autoConfigureActuators: false
   ```

Note

The default values for both platform level and workload level configuration is false.

To run Application Live View with Spring Boot apps, the Spring Boot convention recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Activates the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot`: Exposes the framework flavor of the app

To run Application Live View with Spring Cloud Gateway apps, Spring Boot conventions recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Activates the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot,spring-cloud-gateway`: Exposes the framework flavors of the app

Enable Application Live View for Spring Boot applications

To run Application Live View for Spring Boot apps, Spring Boot conventions recognizes PodIntents and automatically adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Enables the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot`: Exposes the framework flavor of the app

To run Application Live View for Spring Cloud Gateway apps, Spring Boot conventions recognizes PodIntents and adds the following metadata labels:

- `tanzu.app.live.view: "true"`: Enables the connector to observe application pod
- `tanzu.app.live.view.application.name: APPLICATION-NAME`: Identifies the app name to be used internally by Application Live View
- `tanzu.app.live.view.application.actuator.port: ACTUATOR-PORT`: Identifies the port on the pod at which the actuators are available for Application Live View
- `tanzu.app.live.view.application.flavours: spring-boot,spring-cloud-gateway`: Exposes the framework flavors of the app

These metadata labels allow Application Live View to identify pods that are enabled for Application Live View. The metadata labels also tell the Application Live View connector what kind of app it is and on which port the actuators are accessible for Application Live View. For more information, see Configuring and accessing Spring Boot actuators in Tanzu Application Platform.

Verify the applied labels and annotations

To verify the applied labels and annotations, run:

```
kubectl get podintents.conventions.carto.run WORKLOAD-NAME -o yaml
```

Where `WORKLOAD-NAME` is the name of the deployed workload. For example: `tanzu-java-web-app`.

Expected output of Spring Boot workload:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2022-11-14T10:07:55Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-java-web-app
    apps.tanzu.vmware.com/auto-configure-actuators: "true"
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: tanzu-java-web-app
    carto.run/workload-namespace: default
  name: tanzu-java-web-app
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
```

```
    controller: true
    kind: Workload
    name: tanzu-java-web-app
    uid: dfd3c0c2-9d1f-4231-9390-3e16f23bb62d
  resourceVersion: "444497"
  uid: 224de2aa-307a-48e3-a826-2c474c435bb2
spec:
  serviceAccountName: default
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-java-web-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-def
ault@sha256:444686bb8bfbaba5552676140619b00f43c8f85b6823b87676c0ccdcdead65ac
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2022-11-14T10:07:59Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2022-11-14T10:07:59Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        boot.spring.io/actuator: http://:8081/actuator
        boot.spring.io/version: 2.7.3
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/auto-configure-actuators-check
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
          spring-boot-convention/spring-boot-actuator-probes
          spring-boot-convention/app-live-view-appflavour-check
          spring-boot-convention/app-live-view-connector-boot
          spring-boot-convention/app-live-view-appflavours-boot
          appliveview-sample/app-live-view-appflavour-check
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-java-web-app
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-java-web-app
        conventions.carto.run/framework: spring-boot
```

```
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.actuator.path: actuator
        tanzu.app.live.view.application.actuator.port: "8081"
        tanzu.app.live.view.application.flavours: spring-boot
        tanzu.app.live.view.application.name: tanzu-java-web-app
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.endpoint.health.show-details="always"
            -Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.endpoints.we
b.exposure.include="*"
            -Dmanagement.health.probes.enabled="true" -Dmanagement.server.port="8081"
            -Dserver.port="8080" -Dserver.shutdown.grace-period="24s"
        image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-java-web-app-def
ault@sha256:444686bb8bfbaba5552676140619b00f43c8f85b6823b87676c0ccdcdead65ac
        livenessProbe:
          httpGet:
            path: /livez
            port: 8080
            scheme: HTTP
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
```

Expected output of Spring Cloud Gateway workload:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  creationTimestamp: "2022-11-14T10:29:51Z"
  generation: 1
  labels:
    app.kubernetes.io/component: intent
    app.kubernetes.io/part-of: tanzu-scg-web-app
    apps.tanzu.vmware.com/auto-configure-actuators: "true"
    apps.tanzu.vmware.com/workload-type: web
    carto.run/cluster-template-name: convention-template
    carto.run/resource-name: config-provider
    carto.run/supply-chain-name: source-to-url
    carto.run/template-kind: ClusterConfigTemplate
    carto.run/workload-name: tanzu-scg-web-app
    carto.run/workload-namespace: default
  name: tanzu-scg-web-app
  namespace: default
  ownerReferences:
  - apiVersion: carto.run/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Workload
    name: tanzu-scg-web-app
    uid: 5d8cdc5b-0236-471d-8c1e-335e659f1ae6
  resourceVersion: "475756"
  uid: d086f02c-6ff0-47f8-8dee-4da8748d8adc
spec:
```

```
  serviceAccountName: default
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-scg-web-app
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-scg-web-app
    spec:
      containers:
      - image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-scg-web-app-defa
ult@sha256:7656f4ca56b7d0d6376b374643d6ac09c8cdcdbcc13d065f9224651b12724d0b
        name: workload
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
status:
  conditions:
  - lastTransitionTime: "2022-11-14T10:29:58Z"
    message: ""
    reason: Applied
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "2022-11-14T10:29:58Z"
    message: ""
    reason: ConventionsApplied
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "1"
        boot.spring.io/actuator: http://:8081/actuator
        boot.spring.io/version: 2.6.3
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/auto-configure-actuators-check
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
          spring-boot-convention/spring-boot-actuator-probes
          spring-boot-convention/app-live-view-appflavour-check
          spring-boot-convention/app-live-view-connector-boot
          spring-boot-convention/app-live-view-appflavours-boot
          spring-boot-convention/app-live-view-connector-scg
          spring-boot-convention/app-live-view-appflavours-scg
          appliveview-sample/app-live-view-appflavour-check
        developer.conventions/target-containers: workload
      labels:
        app.kubernetes.io/component: run
        app.kubernetes.io/part-of: tanzu-scg-web-app
        apps.tanzu.vmware.com/auto-configure-actuators: "true"
        apps.tanzu.vmware.com/workload-type: web
        carto.run/workload-name: tanzu-scg-web-app
        conventions.carto.run/framework: spring-boot
        tanzu.app.live.view: "true"
        tanzu.app.live.view.application.actuator.path: actuator
        tanzu.app.live.view.application.actuator.port: "8081"
        tanzu.app.live.view.application.flavours: spring-boot_spring-cloud-gateway
        tanzu.app.live.view.application.name: tanzu-scg-web-app
    spec:
```

```
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.endpoint.health.show-details="always"
            -Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.endpoints.we
b.exposure.include="*"
            -Dmanagement.health.probes.enabled="true" -Dmanagement.server.port="8081"
            -Dserver.port="8080"
        image: dev.registry.tanzu.vmware.com/app-live-view/test/tanzu-scg-web-app-defa
ult@sha256:7656f4ca56b7d0d6376b374643d6ac09c8cdcdbcc13d065f9224651b12724d0b
        livenessProbe:
          httpGet:
            path: /livez
            port: 8080
            scheme: HTTP
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
        resources: {}
        securityContext:
          runAsUser: 1000
      serviceAccountName: default
```

List of Spring Boot conventions

This topic tells you about what the conventions do and how to apply them.

When submitting the following pod `Pod Intent` on each convention, the output can change depending on the applied convention.

Before any Spring Boot conventions are applied, the pod intent looks similar to this YAML:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  name: spring-sample
spec:
  template:
    spec:
      containers:
      - name: workload
        image: springio/petclinic
```

Most of the Spring Boot conventions either edit or add properties to the environment variable `JAVA_TOOL_OPTIONS`. You can override those conventions by providing the `JAVA_TOOL_OPTIONS` value you want through the Tanzu CLI or `workload.yaml`.

When a `JAVA_TOOL_OPTIONS` property already exists for a workload, the convention uses the existing value rather than the value that the convention applies by default. The property value that you provide is used for the pod specification mutation.

Set a `JAVA_TOOL_OPTIONS` property for a workload

Do one of the following actions to set `JAVA_TOOL_OPTIONS` property and values:

Use the Tanzu CLI apps plug-in

When creating or updating a workload, set a `JAVA_TOOL_OPTIONS` property using the `--env` flag by running:

```
tanzu apps workload create APP-NAME --env JAVA_TOOL_OPTIONS="-DPROPERTY-NAME=VALUE"
```

For example, to set the management port to `8080` rather than the spring-boot-actuator-convention default port `8081`, run:

```
tanzu apps workload create APP-NAME --env JAVA_TOOL_OPTIONS="-Dmanagement.server.port=8080"
```

Use workload.yaml

Follow these steps:

1. Provide one or more values for the `JAVA_TOOL_OPTIONS` property in the `workload.yaml`. For example:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   ...
   spec:
   env:
   - name: JAVA_TOOL_OPTIONS
     value: -Dmanagement.server.port=8082
   source:
   ...
   ```

2. Apply the `workload.yaml` file by running the command:

   ```
   tanzu apps workload create -f workload.yaml
   ```

Spring Boot convention

If the `spring-boot` dependency is in the metadata within the `SBOM` file under `dependencies`, the Spring Boot convention is applied to the `PodTemplateSpec` object.

The Spring Boot convention adds a label (`conventions.carto.run/framework: spring-boot`) to the `PodTemplateSpec` that describes the framework associated with the workload, and adds an annotation (`boot.spring.io/version: VERSION-NO`) that describes the Spring Boot version of the dependency.

The label and annotation are added for informational purposes only.

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
 annotations:
   kubectl.kubernetes.io/last-applied-configuration: |
     {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"an
notations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...

status:
 conditions:
```

```
  - lastTransitionTime: "..." # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "..."
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/version: 2.3.3.RELEASE
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/spring-boot
      labels:
        conventions.carto.run/framework: spring-boot
    spec:
      containers:
      - image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        resources: {}
```

Spring boot graceful shut down convention

If any of the following dependencies are in the metadata within the `SBOM` file under `dependencies`, the Spring Boot graceful shut down convention is applied to the `PodTemplateSpec` object:

- `spring-boot-starter-tomcat`
- `spring-boot-starter-jetty`
- `spring-boot-starter-reactor-netty`
- `spring-boot-starter-undertow`
- `tomcat-embed-core`

The Graceful Shutdown convention `spring-boot-graceful-shutdown` adds a property in the environment variable `JAVA_TOOL_OPTIONS` with the key `server.shutdown.grace-period`. The key value is calculated to be 80% of the value set in `.target.Spec.TerminationGracePeriodSeconds`. The default value for `.target.Spec.TerminationGracePeriodSeconds` is 30 seconds.

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"a
nnotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...

status:
  conditions:
  - lastTransitionTime: "..." # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "..."
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
```

```
      annotations:
        boot.spring.io/version: 2.3.3.RELEASE
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-graceful-shutdown
      labels:
        conventions.carto.run/framework: spring-boot
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dserver.shutdown.grace-period="24s"
        image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        resources: {}
```

Spring Boot web convention

If any of the following dependencies are in the metadata within the `SBOM` file under `dependencies`, the Spring Boot web convention is applied to the `PodTemplateSpec` object:

- spring-boot
- spring-boot-web

The web convention `spring-boot-web` obtains the `server.port` property from the `JAVA_TOOL_OPTIONS` environment variable and sets it as a port in the `PodTemplateSpec`. If `JAVA_TOOL_OPTIONS` environment variable does not contain a `server.port` property or value, the convention adds the property and sets the value to `8080`, which is the Spring Boot default.

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"a
nnotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...

status:
  conditions:
  - lastTransitionTime: "..." # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "..."
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/version: 2.3.3.RELEASE
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
      labels:
        conventions.carto.run/framework: spring-boot
    spec:
      containers:
      - env:
```

```
        - name: JAVA_TOOL_OPTIONS
          value: -Dserver.port="8080"
        image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        ports:
        - containerPort: 8080
          protocol: TCP
        resources: {}
```

Spring Boot Actuator convention

If the `spring-boot-actuator` dependency is in the metadata within the `SBOM` file under `dependencies`, the Spring Boot actuator convention is applied to the `PodTemplateSpec` object.

The Spring Boot Actuator convention does the following actions:

If the workload-level or platform-level automatic configuration of actuators is enabled:

1. Sets the management port in the `JAVA_TOOL_OPTIONS` environment variable to `8081`.
2. Sets the base path in the `JAVA_TOOL_OPTIONS` environment variable to `/actuator`.
3. Adds an annotation, `boot.spring.io/actuator`, to where the actuator is accessed.

The management port is set to port `8081` for security reasons. Although you can prevent public access to the actuator endpoints that are exposed on the management port when it is set to the default `8080`, the threat of exposure through internal access remains. The best practice for security is to set the management port to something other than `8080`.

However, if a management port number value is provided using the `-Dmanagement.server.port` property in `JAVA_TOOL_OPTIONS`, the Spring Boot actuator convention uses that value rather than its default `8081` as the management port.

You can access the management context of a Spring Boot application by creating a service pointing to port `8081` and base path `/actuator`.

Important

To override the management port setting applied by this convention, see How to set a JAVA_TOOL_OPTIONS property for a workload earlier in this topic. Any alternative methods for setting the management port are overwritten. For example, if you configure the management port using `application.properties/yml` or `config server`, the Spring Boot Actuator convention overrides your configuration.

If the workload-level or platform-level automatic configuration of actuators is deactivated, the Spring Boot actuator convention does not set any `JAVA_TOOLS_OPTIONS` and does not set the annotation `boot.spring.io/actuator`.

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
 annotations:
   kubectl.kubernetes.io/last-applied-configuration: |
     {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"an
notations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...
```

```
status:
 conditions:
 - lastTransitionTime: "..." # This status indicates that all worked as expected
   status: "True"
   type: ConventionsApplied
 - lastTransitionTime: "..."
   status: "True"
   type: Ready
 observedGeneration: 1
 template:
   metadata:
     annotations:
       boot.spring.io/actuator: http://:8081/actuator
       boot.spring.io/version: 2.3.3.RELEASE
       conventions.carto.run/applied-conventions: |-
         spring-boot-convention/spring-boot
         spring-boot-convention/spring-boot-web
         spring-boot-convention/spring-boot-actuator
     labels:
       conventions.carto.run/framework: spring-boot
   spec:
     containers:
     - env:
       - name: JAVA_TOOL_OPTIONS
         value: Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.server.po
rt="8081" -Dserver.port="8080"
       image: index.docker.io/springio/petclinic@sha256:...
       name: workload
       ports:
       - containerPort: 8080
         protocol: TCP
       resources: {}
```

Spring Boot Actuator Probes convention

The Spring Boot Actuator Probes convention is applied only if all of the following conditions are met:

- The `spring-boot-actuator` dependency exists and is **>= 2.6**
- The `JAVA_TOOL_OPTIONS` environment variable does not include the following properties or, if either of the properties is included, it is set to a value of `true`:
    - `-Dmanagement.health.probes.enabled`
    - `-Dmanagement.endpoint.health.probes.add-additional-paths`

The Spring Boot Actuator Probes convention does the following actions:

1. Uses the main server port, which is the `server.port` value on `JAVA_TOOL_OPTIONS`, to set the liveness and readiness probes. For more information see the Kubernetes documentation
2. Adds the following properties and values to the `JAVA_TOOL_OPTIONS` environment variable:
    - `-Dmanagement.health.probes.enabled="true"`
    - `-Dmanagement.endpoint.health.probes.add-additional-paths="true"`

    When this convention is applied, the probes are exposed as follows:
    - Liveness probe: `/livez`
    - Readiness probe: `/readyz`

Example of PodIntent after applying the convention:

```
apiVersion: conventions.carto.run/v1alpha1
kind: PodIntent
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":{"a
nnotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"spe
c":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}

...

status:
  conditions:
  - lastTransitionTime: "..." # This status indicates that all worked as expected
    status: "True"
    type: ConventionsApplied
  - lastTransitionTime: "..."
    status: "True"
    type: Ready
  observedGeneration: 1
  template:
    metadata:
      annotations:
        boot.spring.io/actuator: http://:8080/actuator
        boot.spring.io/version: 2.6.0
        conventions.carto.run/applied-conventions: |-
          spring-boot-convention/spring-boot
          spring-boot-convention/spring-boot-web
          spring-boot-convention/spring-boot-actuator
      labels:
        conventions.carto.run/framework: spring-boot
    spec:
      containers:
      - env:
        - name: JAVA_TOOL_OPTIONS
          value: -Dmanagement.endpoint.health.probes.add-additional-paths="true" -Dman
agement.endpoints.web.base-path="/actuator" -Dmanagement.health.probes.enabled="true"
-Dmanagement.server.port="8081" -Dserver.port="8080"
        image: index.docker.io/springio/petclinic@sha256:...
        name: workload
        livenessProbe:
          httpGet:
            path: /livez
            port: 8080
            scheme: HTTP
        ports:
        - containerPort: 8080
          protocol: TCP
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8080
            scheme: HTTP
        resources: {}
```

Service intent conventions

The Service intent conventions do not change the behavior of the final deployment, but you can use them as added information to process in the supply chain. For example, when an app requires to be bound to database service. This convention adds an annotation and a label to the `PodTemplateSpec` for each detected dependency. It also adds an annotation and a label to the `conventions.carto.run/applied-conventions`.

The list of the supported intents are:

MySQL

- **Name**: `service-intent-mysql`
- **Label**: `services.conventions.apps.tanzu.vmware.com/mysql`
- **Dependencies**: `mysql-connector-java`, `r2dbc-mysql`

PostgreSQL

- **Name**: `service-intent-postgres`
- **Label**: `services.conventions.apps.tanzu.vmware.com/postgres`
- **Dependencies**: `postgresql`, `r2dbc-postgresql`

MongoDB

- **Name**: `service-intent-mongodb`
- **Label**: `services.conventions.apps.tanzu.vmware.com/mongodb`
- **Dependencies**: `mongodb-driver-core`

RabbitMQ

- **Name**: `service-intent-rabbitmq`
- **Label**: `services.conventions.apps.tanzu.vmware.com/rabbitmq`
- **Dependencies**: `amqp-client`

Redis

- **Name**: `service-intent-redis`
- **Label**: `services.conventions.apps.tanzu.vmware.com/redis`
- **Dependencies**: `jedis`

Kafka

- **Name**: `service-intent-kafka`
- **Label**: `services.conventions.apps.tanzu.vmware.com/kafka`
- **Dependencies**: `kafka-clients`

Kafka-streams

- **Name**: `service-intent-kafka-streams`
- **Label**: `services.conventions.apps.tanzu.vmware.com/kafka-streams`
- **Dependencies**: `kafka-streams`

Example

When you apply the `Pod Intent` and the image contains a dependency, for example, of MySQL, then the output of the convention is:

```
  apiVersion: conventions.carto.run/v1alpha1
  kind: PodIntent
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"conventions.carto.run/v1alpha1","kind":"PodIntent","metadata":
{"annotations":{},"name":"spring-sample","namespace":"default"},"spec":{"template":{"s
pec":{"containers":[{"image":"springio/petclinic","name":"workload"}]}}}}
    creationTimestamp: "..."
```

```
    generation: 1
    name: spring-sample
    namespace: default
    resourceVersion: "..."
    uid: ...
  spec:
    serviceAccountName: default
    template:
      metadata: {}
      spec:
        containers:
        - image: springio/petclinic
          name: workload
          resources: {}
  status:
    conditions:
    - lastTransitionTime: "..." # This status indicates that all worked as expected
      status: "True"
      type: ConventionsApplied
    - lastTransitionTime: "..."
      status: "True"
      type: Ready
    observedGeneration: 1
    template:
      metadata:
        annotations:
          boot.spring.io/actuator: http://:8080/actuator
          boot.spring.io/version: 2.3.3.RELEASE
          conventions.carto.run/applied-conventions: |-
            spring-boot-convention/spring-boot
            spring-boot-convention/spring-boot-web
            spring-boot-convention/spring-boot-actuator
            spring-boot-convention/service-intent-mysql
          services.conventions.apps.tanzu.vmware.com/mysql: mysql-connector-java/8.0.2
1
        labels:
          conventions.apps.tanzu.vmware.com/framework: spring-boot
          services.conventions.apps.tanzu.vmware.com/mysql: workload
      spec:
        containers:
        - env:
          - name: JAVA_TOOL_OPTIONS
            value: Dmanagement.endpoints.web.base-path="/actuator" -Dmanagement.serve
r.port="8081" -Dserver.port="8080"
          image: index.docker.io/springio/petclinic@sha256:...
          name: workload
          ports:
          - containerPort: 8080
            protocol: TCP
          resources: {}
```

Troubleshoot Spring Boot conventions

This topic tells you how to troubleshoot Spring Boot conventions.

Collect logs

If you have trouble, you can retrieve and examine logs from the Spring Boot convention server as follows:

1. The Spring Boot convention server creates a namespace to contain all of the associated resources. By default the namespace is `spring-boot-convention`. To inspect the logs, run:

```
kubectl logs -l app=spring-boot-webhook -n spring-boot-convention
```

For example:

```
$ kubectl logs -l app=spring-boot-webhook -n spring-boot-convention

{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
nventions/server.go:83","msg":"Successfully applied convention: spring-boot","c
omponent":"spring-boot-conventions"}
{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
nventions/server.go:83","msg":"Successfully applied convention: spring-boot-gra
ceful-shutdown","component":"spring-boot-conventions"}
{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
nventions/server.go:83","msg":"Successfully applied convention: spring-boot-we
b","component":"spring-boot-conventions"}
{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
nventions/server.go:83","msg":"Successfully applied convention: spring-boot-act
uator","component":"spring-boot-conventions"}
{"level":"info","timestamp":"2021-11-11T16:00:26.597Z","caller":"spring-boot-co
nventions/server.go:83","msg":"Successfully applied convention: service-intent-
mysql","component":"spring-boot-conventions"}
```

2. For all of the conventions that were applied successfully, a log entry is added. If an error occurs, a log entry is added with a description.

Overview of Spring Cloud Gateway for Kubernetes

Spring Cloud Gateway for Kubernetes is an API gateway solution based on the open-source Spring Cloud Gateway project. It provides a simple means to route internal or external API requests to application services that expose APIs.

Spring Cloud Gateway for Kubernetes handles cross-cutting concerns on behalf of API development teams, including single sign-on (SSO), access control, rate limiting, resiliency, security, and more.

It enables you to accelerate API delivery using modern cloud-native patterns with any programming language you choose for API development. It also integrates with your existing CI/CD pipeline strategy.

For more information about Spring Cloud Gateway for Kubernetes, see the Spring Cloud Gateway for Kubernetes documentation.

Overview of Spring Cloud Gateway for Kubernetes

Spring Cloud Gateway for Kubernetes is an API gateway solution based on the open-source Spring Cloud Gateway project. It provides a simple means to route internal or external API requests to application services that expose APIs.

Spring Cloud Gateway for Kubernetes handles cross-cutting concerns on behalf of API development teams, including single sign-on (SSO), access control, rate limiting, resiliency, security, and more.

It enables you to accelerate API delivery using modern cloud-native patterns with any programming language you choose for API development. It also integrates with your existing CI/CD pipeline strategy.

For more information about Spring Cloud Gateway for Kubernetes, see the Spring Cloud Gateway for Kubernetes documentation.

Install Spring Cloud Gateway for Kubernetes

This topic describes how to install Spring Cloud Gateway for Kubernetes from the Tanzu Application Platform package repository.

Prerequisites

Before installing Spring Cloud Gateway, complete all prerequisites for installing Tanzu Application Platform. For more information, see Prerequisites.

Install

To install Spring Cloud Gateway:

1. See which versions of Spring Cloud Gateway are available to install from the Tanzu Application Platform repository by running:

   ```
   tanzu package available list spring-cloud-gateway.tanzu.vmware.com --namespace
   tap-install
   ```

   For example:

   ```
   $ tanzu package available list spring-cloud-gateway.tanzu.vmware.com --namespac
   e tap-install

   NAME                                    VERSION  RELEASED-AT
   spring-cloud-gateway.tanzu.vmware.com  2.0.0    2022-02-01T00:00:00Z
   ```

2. (Optional) View the changes you can make to the default installation settings by running:

   ```
   tanzu package available get spring-cloud-gateway.tanzu.vmware.com/VERSION-NUMBE
   R \
     --namespace tap-install --values-schema
   ```

   Where `VERSION-NUMBER` is the version of the package listed earlier.

   For example:

   ```
   tanzu package available get spring-cloud-gateway.tanzu.vmware.com/2.0.0 \
     --namespace tap-install --values-schema
   ```

   You can use the information to generate a values override file for use in the following installation step.

   For more information about values schema options, see the Spring Cloud Gateway for Kubernetes documentation.

   Important

   The value of `deployment.namespace` must always be set to the same value as the `--namespace` flag.

3. Install Spring Cloud Gateway by running:

   Default values
   Run this command to install Spring Cloud Gateway with the default values

   ```
   tanzu package install spring-cloud-gateway \
     --package spring-cloud-gateway.tanzu.vmware.com \
   ```

```
  --version VERSION-NUMBER \
  --namespace tap-install
```

For example:

```
$ tanzu package install spring-cloud-gateway \
    --package spring-cloud-gateway.tanzu.vmware.com \
    --version 2.0.0 \
    --namespace tap-install

Installing package 'spring-cloud-gateway.tanzu.vmware.com'
Getting package metadata for 'spring-cloud-gateway.tanzu.vmware.com'
Creating service account 'spring-cloud-gateway-tap-install-sa'
Creating cluster admin role 'spring-cloud-gateway-tap-install-cluster-role'
Creating cluster role binding 'spring-cloud-gateway-tap-install-cluster-roleb
inding'
Creating package resource
Waiting for 'PackageInstall' reconciliation for 'spring-cloud-gateway'
'PackageInstall' resource install status: Reconciling
'PackageInstall' resource install status: ReconcileSucceeded

Added installed package 'spring-cloud-gateway'
```

Overriding values

Run this command to install Spring Cloud Gateway while overriding the default values

```
tanzu package install spring-cloud-gateway \
  --package spring-cloud-gateway.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file values.yml
```

Overview of Supply Chain Choreographer for Tanzu

This topic introduces you to Supply Chain Choreographer.

Overview

Supply Chain Choreographer is based on open source Cartographer. It allows App Operators to create pre-approved paths to production by integrating Kubernetes resources with the elements of their existing toolchains, for example, Jenkins.

Each pre-approved supply chain creates a path to production. Orchestrating supply chain resources including, test, build, scan, and deploy allows developers to focus on delivering value to their users and provides App Operators the assurance that all code in production has passed through all the steps of an approved workflow.

Out of the Box Supply Chains

Out of the box supply chains are provided with Tanzu Application Platform.

The following three supply chains are included:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain with Testing
- Out of the Box Supply Chain with Testing and Scanning

As auxiliary components, Tanzu Application Platform also includes:

- Out of the Box Templates, for providing templates used by the supply chains to perform common tasks such as fetching source code, running tests, and building container images.
- Out of the Box Delivery Basic, for delivering to a Kubernetes cluster the configuration built throughout a supply chain

Both Templates and Delivery Basic are requirements for the Supply Chains.

Supply Chain Choreographer supports the following pipeline types:

- Tekton pipelines
- Jenkins pipelines

Overview of Supply Chain Choreographer for Tanzu

This topic introduces you to Supply Chain Choreographer.

Overview

Supply Chain Choreographer is based on open source Cartographer. It allows App Operators to create pre-approved paths to production by integrating Kubernetes resources with the elements of their existing toolchains, for example, Jenkins.

Each pre-approved supply chain creates a path to production. Orchestrating supply chain resources including, test, build, scan, and deploy allows developers to focus on delivering value to their users and provides App Operators the assurance that all code in production has passed through all the steps of an approved workflow.

Out of the Box Supply Chains

Out of the box supply chains are provided with Tanzu Application Platform.

The following three supply chains are included:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain with Testing
- Out of the Box Supply Chain with Testing and Scanning

As auxiliary components, Tanzu Application Platform also includes:

- Out of the Box Templates, for providing templates used by the supply chains to perform common tasks such as fetching source code, running tests, and building container images.
- Out of the Box Delivery Basic, for delivering to a Kubernetes cluster the configuration built throughout a supply chain

Both Templates and Delivery Basic are requirements for the Supply Chains.

Supply Chain Choreographer supports the following pipeline types:

- Tekton pipelines
- Jenkins pipelines

Install Supply Chain Choreographer

This topic describes how you can install Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Choreographer. For more information about profiles, see Components and installation profiles..

The Supply Chain Choreographer is now bundled with the Cartographer Conventions. For information on configuring and using Cartographer Conventions, see Creating conventions.

Supply Chain Choreographer provides the custom resource definitions the supply chain uses. Each pre-approved supply chain creates a clear road to production and orchestrates supply chain resources. You can test, build, scan, and deploy. Developers can focus on delivering value to users. Application operators can rest assured that all code in production has passed through an approved workflow.

For example, Supply Chain Choreographer passes the results of fetching source code to the component that builds a container image of it, and then passes the container image to a component that deploys the image.

Prerequisites

Before installing Supply Chain Choreographer:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

Install

To install Supply Chain Choreographer:

1. Get the values schema to see what properties can be configured during installation. Run:

```
tanzu package available get cartographer.tanzu.vmware.com/0.7.1+tap.1 --values-
schema --namespace tap-install

KEY                                     DEFAULT  TYPE     DESCRIPTION
ca_cert_data                            ""       string   Optional: PEM Encode
d certificate data for image registries with private CA.
cartographer.concurrency.max_deliveries  2        integer  Optional: maximum nu
mber of Deliverables to process concurrently.
cartographer.concurrency.max_runnables   2        integer  Optional: maximum nu
mber of Runnables to process concurrently.
cartographer.concurrency.max_workloads   2        integer  Optional: maximum nu
mber of Workloads to process concurrently.
cartographer.resources.limits.cpu        1000m            Optional: maximum am
ount of cpu resources to allow the controller to use
cartographer.resources.limits.memory     128Mi            Optional: maximum am
ount of memory to allow the controller to use
cartographer.resources.requests.cpu      250m             Optional: minimum am
ount of cpu to reserve
cartographer.resources.requests.memory   128Mi            Optional: minimum am
ount of memory to reserve
conventions.resources.limits.cpu         1000m            Optional: maximum am
ount of cpu resources to allow the controller to use
conventions.resources.limits.memory      128Mi            Optional: maximum am
ount of memory to allow the controller to use
conventions.resources.requests.cpu       250m             Optional: minimum am
ount of cpu to reserve
conventions.resources.requests.memory    128Mi            Optional: minimum am
ount of memory to reserve
excluded_components                              array    Optional: List of co
mponents to exclude from installation (e.g. [conventions])
```

```
aws_iam_role_arn                          ""       string   Optional: Arn role t
hat has access to pull images from ECR container registry
```

2. Install v0.4.0 of the `cartographer.tanzu.vmware.com` package, naming the installation `cartographer`. Run:

   ```
   tanzu package install cartographer \
     --namespace tap-install \
     --package cartographer.tanzu.vmware.com \
     --version 0.7.1+tap.1
   ```

   Example output:

   ```
   | Installing package 'cartographer.tanzu.vmware.com'
   | Getting namespace 'tap-install'
   | Getting package metadata for 'cartographer.tanzu.vmware.com'
   | Creating service account 'cartographer-tap-install-sa'
   | Creating cluster admin role 'cartographer-tap-install-cluster-role'
   | Creating cluster role binding 'cartographer-tap-install-cluster-rolebinding'
   - Creating package resource
   \ Package install status: Reconciling

   Added installed package 'cartographer' in namespace 'tap-install'
   ```

Out of the Box Supply Chain Basic for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain Basic for Supply Chain Choreographer.

This package contains Cartographer supply chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains the most basic supply chains that focus on providing a quick path to deployment making no use of testing or scanning resources.

The supply chains in this package perform the following:

- Building from source code:
    1. Watching a Git repository, Maven repository, or local directory for changes
    2. Building a container image out of the source code with Buildpacks
    3. Applying operator-defined conventions to the container definition
    4. Creating a deliverable object for deploying the application to a cluster
    5. (Experimental) Alternatively, outputting a Carvel Package containing the application to a Git Repository
- Using a prebuilt application image:
    1. Applying operator-defined conventions to the container definition
    2. Creating a deliverable object for deploying the application to a cluster
    3. (Experimental) Alternatively, outputting a Carvel Package containing the application to a Git Repository

Prerequisites

To use this package, you must:

- Install Out of the Box Templates.
- Configure the Developer namespace with auxiliary objects that are used by the supply chain as described in the following section.
- (Optionally) install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

Developer Namespace

The supply chains provide definitions of many of the objects that they create to transform the source code to a container image and make it available as an application in a cluster.

The developer must provide or configure particular objects in the developer namespace so that the supply chain can provide credentials and use permissions granted to a specific development team.

The objects that the developer must provide or configure include:

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

Registries Secrets

Regardless of the supply chain that a workload goes through, there must be Kubernetes secrets in the developer namespace containing credentials for both pushing and pulling the container image that the supply chain builds when source code is provided. The developer namespace must also contain registry credentials for Kubernetes to run pods using images from the installation of Tanzu Application Platform.

1. Add read/write registry credentials for pushing and pulling application images:

   ```
   tanzu secret registry add registry-credentials \
     --server REGISTRY-SERVER \
     --username REGISTRY-USERNAME \
     --password REGISTRY-PASSWORD \
     --namespace YOUR-NAMESPACE
   ```

   Where:

   - `YOUR-NAMESPACE` is the name you want to use for the developer namespace. For example, use `default` for the default namespace.
   - `REGISTRY-SERVER` is the URL of the registry. For Docker Hub, this must be `https://index.docker.io/v1/`. Specifically, it must have the leading `https://`, the `v1` path, and the trailing `/`. For Google Container Registry (GCR), this is `gcr.io`. Based on the information used in Installing the Tanzu Application Platform package and profiles, you can use the same registry server as in `ootb_supply_chain_basic - registry - server`.

2. Add a placeholder secret for gathering the credentials used for pulling container images from the installation of Tanzu Application Platform:

   ```
   cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
   apiVersion: v1
   kind: Secret
   ```

```
metadata:
  name: tap-registry
  annotations:
    secretgen.carvel.dev/image-pull-secret: ""
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: e30K
EOF
```

With the two secrets created:

- `tap-registry` is a placeholder secret filled indirectly by `secretgen-controller` Tanzu Application Platform credentials set up during the installation of Tanzu Application Platform.
- `registry-credentials` is a secret providing credentials for the registry where application container images are pushed to.

The following section discusses setting up the identity required for the workload.

ServiceAccount

In a Kubernetes cluster, a ServiceAccount provides a way of representing an actor within the Kubernetes role-based access control (RBAC) system. In the case of a developer namespace, this represents a developer or development team.

You can directly attach secrets to the ServiceAccount through both the `secrets` and `imagePullSecets` fields. This allows you to provide indirect ways for resources to find credentials without knowing the exact name of the secrets.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

Important

The ServiceAccount must have the secrets created earlier linked to it. If it does not, services like Tanzu Build Service (used in the supply chain) lack the necessary credentials for pushing the images it builds for that workload.

RoleBinding

As the supply chain takes action in the cluster on behalf of the users who created the workload, it needs permissions within Kubernetes' RBAC system to do so.

Tanzu Application Platform v1.2 includes two ClusterRoles that describe all of the necessary permissions to grant to the service account:

- `workload` clusterrole, providing the necessary roles for the supply chains to manage the resources prescribed by them.
- `deliverable` clusterrole, providing the roles for deliveries to deploy to the cluster the application Kubernetes objects produced by the supply chain.

To provide those permissions to the identity you created for this workload, bind the `workload` ClusterRole to the ServiceAccount you created above:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-workload
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: workload
subjects:
  - kind: ServiceAccount
    name: default
```

If this is a Build cluster, and you do not intend to run the application in it, this single RoleBinding is all that's necessary.

If you intend to also deploy the application that's been built, bind to the same ServiceAccount the `deliverable` ClusterRole too:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-deliverable
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: deliverable
subjects:
  - kind: ServiceAccount
    name: default
```

For more information about authentication and authorization in Tanzu Application Platform, see Overview of Default roles for Tanzu Application Platform.

Developer workload

With the developer namespace set up with the preceding objects, such as secret, serviceaccount, and rolebinding, you can create the workload object.

To do so, use the `apps` plug-in from the Tanzu CLI:

```
tanzu apps workload create FLAGS WORKLOAD-NAME
```

Where:

- `FLAGS` are the one or more flags you want to include.
- `WORKLOAD-NAME` is the name of the workload you want to target.

Depending on what you are aiming to achieve, you can set different flags. To know more about those (including details about different features of the supply chains), see the following sections:

- Building from source, for more information about different ways of creating a workload where the application is built from source code.
- Using an existing image, for more information about how to use prebuilt images in the supply chains.
- GitOps vs RegistryOps, for a description of the different ways of propagating the deployment configuration through external systems (Git repositories and image registries).

- Carvel Package Workflow, for information about how to configure workloads to output Carvel Packages for delivery through Git repositories.

Out of the Box Supply Chain Basic for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain Basic for Supply Chain Choreographer.

This package contains Cartographer supply chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains the most basic supply chains that focus on providing a quick path to deployment making no use of testing or scanning resources.

The supply chains in this package perform the following:

- Building from source code:
    1. Watching a Git repository, Maven repository, or local directory for changes
    2. Building a container image out of the source code with Buildpacks
    3. Applying operator-defined conventions to the container definition
    4. Creating a deliverable object for deploying the application to a cluster
    5. (Experimental) Alternatively, outputting a Carvel Package containing the application to a Git Repository
- Using a prebuilt application image:
    1. Applying operator-defined conventions to the container definition
    2. Creating a deliverable object for deploying the application to a cluster
    3. (Experimental) Alternatively, outputting a Carvel Package containing the application to a Git Repository

Prerequisites

To use this package, you must:

- Install Out of the Box Templates.
- Configure the Developer namespace with auxiliary objects that are used by the supply chain as described in the following section.
- (Optionally) install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

Developer Namespace

The supply chains provide definitions of many of the objects that they create to transform the source code to a container image and make it available as an application in a cluster.

The developer must provide or configure particular objects in the developer namespace so that the supply chain can provide credentials and use permissions granted to a specific development team.

The objects that the developer must provide or configure include:

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.

- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

Registries Secrets

Regardless of the supply chain that a workload goes through, there must be Kubernetes secrets in the developer namespace containing credentials for both pushing and pulling the container image that the supply chain builds when source code is provided. The developer namespace must also contain registry credentials for Kubernetes to run pods using images from the installation of Tanzu Application Platform.

1. Add read/write registry credentials for pushing and pulling application images:

   ```
   tanzu secret registry add registry-credentials \
     --server REGISTRY-SERVER \
     --username REGISTRY-USERNAME \
     --password REGISTRY-PASSWORD \
     --namespace YOUR-NAMESPACE
   ```

   Where:

   - `YOUR-NAMESPACE` is the name you want to use for the developer namespace. For example, use `default` for the default namespace.
   - `REGISTRY-SERVER` is the URL of the registry. For Docker Hub, this must be `https://index.docker.io/v1/`. Specifically, it must have the leading `https://`, the `v1` path, and the trailing `/`. For Google Container Registry (GCR), this is `gcr.io`. Based on the information used in Installing the Tanzu Application Platform package and profiles, you can use the same registry server as in `ootb_supply_chain_basic - registry - server`.

2. Add a placeholder secret for gathering the credentials used for pulling container images from the installation of Tanzu Application Platform:

   ```
   cat <<EOF | kubectl -n YOUR-NAMESPACE apply -f -
   apiVersion: v1
   kind: Secret
   metadata:
     name: tap-registry
     annotations:
       secretgen.carvel.dev/image-pull-secret: ""
   type: kubernetes.io/dockerconfigjson
   data:
     .dockerconfigjson: e30K
   EOF
   ```

With the two secrets created:

- `tap-registry` is a placeholder secret filled indirectly by `secretgen-controller` Tanzu Application Platform credentials set up during the installation of Tanzu Application Platform.
- `registry-credentials` is a secret providing credentials for the registry where application container images are pushed to.

The following section discusses setting up the identity required for the workload.

ServiceAccount

In a Kubernetes cluster, a ServiceAccount provides a way of representing an actor within the Kubernetes role-based access control (RBAC) system. In the case of a developer namespace, this represents a developer or development team.

You can directly attach secrets to the ServiceAccount through both the `secrets` and `imagePullSecets` fields. This allows you to provide indirect ways for resources to find credentials without knowing the exact name of the secrets.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

Important

The ServiceAccount must have the secrets created earlier linked to it. If it does not, services like Tanzu Build Service (used in the supply chain) lack the necessary credentials for pushing the images it builds for that workload.

RoleBinding

As the supply chain takes action in the cluster on behalf of the users who created the workload, it needs permissions within Kubernetes' RBAC system to do so.

Tanzu Application Platform v1.2 includes two ClusterRoles that describe all of the necessary permissions to grant to the service account:

- `workload` clusterrole, providing the necessary roles for the supply chains to manage the resources prescribed by them.
- `deliverable` clusterrole, providing the roles for deliveries to deploy to the cluster the application Kubernetes objects produced by the supply chain.

To provide those permissions to the identity you created for this workload, bind the `workload` ClusterRole to the ServiceAccount you created above:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-workload
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: workload
subjects:
  - kind: ServiceAccount
    name: default
```

If this is a Build cluster, and you do not intend to run the application in it, this single RoleBinding is all that's necessary.

If you intend to also deploy the application that's been built, bind to the same ServiceAccount the `deliverable` ClusterRole too:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: default-permit-deliverable
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: deliverable
subjects:
  - kind: ServiceAccount
    name: default
```

For more information about authentication and authorization in Tanzu Application Platform, see Overview of Default roles for Tanzu Application Platform.

Developer workload

With the developer namespace set up with the preceding objects, such as secret, serviceaccount, and rolebinding, you can create the workload object.

To do so, use the `apps` plug-in from the Tanzu CLI:

```
tanzu apps workload create FLAGS WORKLOAD-NAME
```

Where:

- `FLAGS` are the one or more flags you want to include.
- `WORKLOAD-NAME` is the name of the workload you want to target.

Depending on what you are aiming to achieve, you can set different flags. To know more about those (including details about different features of the supply chains), see the following sections:

- Building from source, for more information about different ways of creating a workload where the application is built from source code.
- Using an existing image, for more information about how to use prebuilt images in the supply chains.
- GitOps vs RegistryOps, for a description of the different ways of propagating the deployment configuration through external systems (Git repositories and image registries).
- Carvel Package Workflow, for information about how to configure workloads to output Carvel Packages for delivery through Git repositories.

Install Out of the Box Supply Chain Basic for Supply Chain Choreographer

This topic shows you how to install the Out of the Box Supply Chain Basic package for Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Supply Chain Basic. For more information about profiles, see Components and installation profiles.

The Out of the Box Supply Chain Basic package provides the most basic ClusterSupplyChain that brings an application from source code to a deployed instance of it running in a Kubernetes environment.

Prerequisites

Fulfill the following prerequisites:

- Fulfill the prerequisites for installing Tanzu Application Platform.
- Install Supply Chain Choreographer.

Install

To install Out of the Box Supply Chain Basic:

1. Familiarize yourself with the set of values of the package that can be configured by running:

```
tanzu package available get ootb-supply-chain-basic.tanzu.vmware.com/0.7.0 \
  --values-schema \
  -n tap-install
```

For example:

```
KEY                                    DESCRIPTION

registry.repository                     Name of the repository in the image regi
stry server where the application
                                        images from the workload should be pushe
d (required).

registry.server                         Name of the registry server where applic
ation images should be pushed to
                                        (required).

git_implementation                      Determines which git client library to u
se. Valid options are go-git or
                                        libgit2.

gitops.server_address                   Default server address to be used for fo
rming Git URLs for pushing
                                        Kubernetes configuration produced by the
supply chain. This must
                                        include the scheme/protocol (e.g. http
s:// or ssh://)

gitops.repository_owner                 Default project or user of the repositor
y. Used to create URLs for pushing
                                        Kubernetes configuration produced by the
supply chain.

gitops.repository_name                  Default repository name used for forming
Git URLs for pushing Kubernetes
                                        configuration produced by the supply cha
in.

gitops.username                         Default user name to be used for the com
mits produced by the supply chain.

gitops.branch                           Default branch to use for pushing Kubern
etes configuration files produced
                                        by the supply chain.

gitops.commit_message                   Default git commit message to write when
publishing Kubernetes
                                        configuration files produces by the supp
ly chain to git.
```

```
gitops.email                              Default user email to be used for the co
mmits produced by the supply chain.

gitops.ssh_secret                         Name of the default Secret containing SS
H credentials to lookup in the
                                          developer namespace for the supply chain
to fetch source code from and
                                          push configuration to.

gitops.commit_strategy                    Specification of how commits are made to
the branch; directly or through a
                                          pull request.

gitops.repository_prefix                  DEPRECATED: Use server_address and repos
itory_owner instead.
                                          Default prefix to be used for forming Gi
t SSH URLs for pushing Kubernetes
                                          configuration produced by the supply cha
in.

gitops.pull_request.server_kind           The git source control platform used

gitops.pull_request.commit_branch         The branch to which commits will be mad
e, before opening a pull request
                                          to the branch specified in .gitops.branc
h If the string "" is specified,
                                          an essentially random string will be use
d for the branch name, in order
                                          to prevent collisions.

gitops.pull_request.pull_request_title The title for the pull request

gitops.pull_request.pull_request_body  Any further information to add to the pu
ll request

cluster_builder                           Name of the Tanzu Build Service ClusterB
uilder to
                                          use by default on image objects managed
by the supply chain.

service_account                           Name of the service account in the names
pace where the Workload
                                          is submitted to utilize for providing re
gistry credentials to
                                          Tanzu Build Service Image objects as wel
l as deploying the
                                          application.

maven.repository.url                      The URL of the Maven repository to be us
ed when pulling Maven
                                          artifacts.  HTTP is not supported.  e.
g.: "https://repo.maven.apache.org/maven"

maven.repository.secret_name              The name of the Secret resource that con
tains the credentials used
                                          to access the Maven repository.
```

2. Create a file named `ootb-supply-chain-basic-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

```
registry:
  server: REGISTRY-SERVER
  repository: REGISTRY-REPOSITORY

gitops:
```

```
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh
  commit_strategy: direct

maven:
  repository:
    url: https://my-maven-repository/releases
    secret_name: my-maven-repository-credentials

cluster_builder: default
service_account: default
```

3. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-basic \
  --package ootb-supply-chain-basic.tanzu.vmware.com \
  --version 0.7.0 \
  --namespace tap-install \
  --values-file ootb-supply-chain-basic-values.yaml
```

Example output:

```
\ Installing package 'ootb-supply-chain-basic.tanzu.vmware.com'
| Getting package metadata for 'ootb-supply-chain-basic.tanzu.vmware.com'
| Creating service account 'ootb-supply-chain-basic-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-basic-tap-install-cluster-rol
e'
| Creating cluster role binding 'ootb-supply-chain-basic-tap-install-cluster-ro
lebinding'
| Creating secret 'ootb-supply-chain-basic-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'ootb-supply-chain-basic'
/ 'PackageInstall' resource install status: Reconciling


 Added installed package 'ootb-supply-chain-basic' in namespace 'tap-install'
```

Out of the Box Supply Chain with Testing for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with Testing for Supply Chain Choreographer.

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It passes the source code forward to image building only if the testing pipeline supplied by the developers runs successfully.

This package includes all the capabilities of the Out of the Box Supply Chain Basic, but adds testing with Tekton.

For workloads that use either source code or prebuilt images, it performs the following:

- Building from source code:
    1. Watching a Git Repository or local directory for changes
    2. Running tests from a developer-provided Tekton pipeline

3. Building a container image out of the source code with Buildpacks
4. Applying operator-defined conventions to the container definition
5. Deploying the application to the same cluster

- Using a prebuilt application image:
    1. Applying operator-defined conventions to the container definition
    2. Creating a deliverable object for deploying the application to a cluster

Prerequisites

To use this supply chain, ensure:

- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing **is installed**.
- Out of the Box Supply Chain With Testing and Scanning **is NOT installed**.
- Developer namespace is configured with the objects per Out of the Box Supply Chain Basic guidance. This supply chain is in addition to the basic one.
- (optionally) Install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

To verify that you have the right set of supply chains installed (that is, the one with Scanning and *not* the one with testing), run:

```
tanzu apps cluster-supply-chain list
```

```
NAME                       LABEL SELECTOR
source-test-to-url         apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/w
orkload-type=web
source-to-url              apps.tanzu.vmware.com/workload-type=web
```

If you see `source-test-scan-to-url` in the list, the setup is wrong: you **must not have the *source-test-scan-to-url* installed** at the same time as *source-test-to-url*.

Developer Namespace

As mentioned in the prerequisites section, this supply chain builds on the previous Out of the Box Supply Chain, so only additions are included here.

To make sure you have configured the namespace correctly, it is important that the namespace has the following objects in it (including the ones marked with '*new*' whose explanation and details are provided below):

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.

  For more information, see Out of the Box Supply Chain Basic.

- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain

  For more information, see Out of the Box Supply Chain Basic.

- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

For more information, see Out of the Box Supply Chain Basic.

- **Tekton pipeline** (*new*): A pipeline runs whenever the supply chain hits the stage of testing the source code.

Below you will find details about the new objects compared to Out of the Box Supply Chain Basic.

Updates to the developer Namespace

For source code testing to be present in the supply chain, a Tekton Pipeline must exist in the same namespace as the Workload so that, at the right moment, the Tekton PipelineRun object that gets created to run the tests can reference such developer-provided Pipeline.

So, aside from the objects previously defined in the Out of the Box Supply Chain Basic section, you need to include one more:

- `tekton/Pipeline`: the definition of a series of tasks to run against the source code that has been found by earlier resources in the Supply Chain.

Tekton/Pipeline

By default, the workload is matched to the corresponding pipeline to run using labels. Pipelines must have the label `apps.tanzu.vmware.com/pipeline: test` at a minimum, but you can add additional labels for granularity. This provides a default match in the event that no other labels are provided. The pipeline expects two parameters:

- `source-url`, an HTTP address where a `.tar.gz` file containing all the source code to be tested can be found
- `source-revision`, the revision of the commit or image reference (in case of `workload.spec.source.image` being set instead of `workload.spec.source.git`)

For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test      # (!) required
spec:
  params:
    - name: source-url                        # (!) required
    - name: source-revision                   # (!) required
  tasks:
    - name: test
      params:
        - name: source-url
          value: $(params.source-url)
        - name: source-revision
          value: $(params.source-revision)
      taskSpec:
        params:
          - name: source-url
          - name: source-revision
        steps:
          - name: test
            image: gradle
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              ./mvnw test
```

At this point, changes to the developer-provided Tekton Pipeline do not automatically trigger a re-run of the pipeline. That is, a new Tekton PipelineRun is not automatically created if a field in the Pipeline object is changed. As a workaround, the latest PipelineRun created can be deleted, which triggers a re-run.

Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script, as shown in the following example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test
spec:
  #...
        steps:
          - name: test
            image: <image_that_has_JDK_and_Go>
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              make test
```

- Update the pipeline resources to include labels that differentiate between the pipelines. For example, differentiate between Java and Go pipelines by adding labels for Java and Go:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: java-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: java
spec:
  #...
        steps:
          - name: test
            image: gradle
            script: |-
              # ...
              ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: go
spec:
  #...
        steps:
          - name: test
            image: golang
            script: |-
```

```
              # ...
              go test -v ./...
```

To match the correct pipeline, you add a `testing_pipeline_matching_labels` parameter to the workload. For example, if you want to match to the Java pipeline, you have the following `workload.yaml`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: sample-java-app
  labels:
    apps.tanzu.vmware.com/has-tests: true
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: sample-java-app
spec:
  params:
    - name: testing_pipeline_matching_labels
      value:
        apps.tanzu.vmware.com/pipeline: test
        apps.tanzu.vmware.com/language: java
  ...
```

This matches the workload to the pipeline with the `apps.tanzu.vmware.com/language: java` label.

Developer Workload

With the Tekton Pipeline object submitted to the same namespace as the one where the Workload will be submitted to, you can submit your Workload.

Regardless of the workflow being targeted (local development or gitops), the Workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload with tests enabled by means of the `has-tests` label.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    git:
     14 + |      ref:
     15 + |        branch: main
     16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     17 + |    subPath: tanzu-java-web-app
```

Out of the Box Supply Chain with Testing for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with Testing for Supply Chain Choreographer.

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It passes the source code forward to image building only if the testing pipeline supplied by the developers runs successfully.

This package includes all the capabilities of the Out of the Box Supply Chain Basic, but adds testing with Tekton.

For workloads that use either source code or prebuilt images, it performs the following:

- Building from source code:
    1. Watching a Git Repository or local directory for changes
    2. Running tests from a developer-provided Tekton pipeline
    3. Building a container image out of the source code with Buildpacks
    4. Applying operator-defined conventions to the container definition
    5. Deploying the application to the same cluster
- Using a prebuilt application image:
    1. Applying operator-defined conventions to the container definition
    2. Creating a deliverable object for deploying the application to a cluster

Prerequisites

To use this supply chain, ensure:

- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing **is installed**.
- Out of the Box Supply Chain With Testing and Scanning **is NOT installed**.
- Developer namespace is configured with the objects per Out of the Box Supply Chain Basic guidance. This supply chain is in addition to the basic one.
- (optionally) Install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

To verify that you have the right set of supply chains installed (that is, the one with Scanning and *not* the one with testing), run:

```
tanzu apps cluster-supply-chain list
```

```
NAME                       LABEL SELECTOR
source-test-to-url         apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/w
orkload-type=web
source-to-url              apps.tanzu.vmware.com/workload-type=web
```

If you see `source-test-scan-to-url` in the list, the setup is wrong: you **must not have the *source-test-scan-to-url* installed** at the same time as *source-test-to-url*.

Developer Namespace

As mentioned in the prerequisites section, this supply chain builds on the previous Out of the Box Supply Chain, so only additions are included here.

To make sure you have configured the namespace correctly, it is important that the namespace has the following objects in it (including the ones marked with '*new*' whose explanation and details are provided below):

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.

  For more information, see Out of the Box Supply Chain Basic.

- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain

  For more information, see Out of the Box Supply Chain Basic.

- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

  For more information, see Out of the Box Supply Chain Basic.

- **Tekton pipeline** (*new*): A pipeline runs whenever the supply chain hits the stage of testing the source code.

Below you will find details about the new objects compared to Out of the Box Supply Chain Basic.

Updates to the developer Namespace

For source code testing to be present in the supply chain, a Tekton Pipeline must exist in the same namespace as the Workload so that, at the right moment, the Tekton PipelineRun object that gets created to run the tests can reference such developer-provided Pipeline.

So, aside from the objects previously defined in the Out of the Box Supply Chain Basic section, you need to include one more:

- `tekton/Pipeline`: the definition of a series of tasks to run against the source code that has been found by earlier resources in the Supply Chain.

Tekton/Pipeline

By default, the workload is matched to the corresponding pipeline to run using labels. Pipelines must have the label `apps.tanzu.vmware.com/pipeline: test` at a minimum, but you can add additional labels for granularity. This provides a default match in the event that no other labels are provided. The pipeline expects two parameters:

- `source-url`, an HTTP address where a `.tar.gz` file containing all the source code to be tested can be found
- `source-revision`, the revision of the commit or image reference (in case of `workload.spec.source.image` being set instead of `workload.spec.source.git`)

For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test      # (!) required
spec:
  params:
    - name: source-url                        # (!) required
```

```
  - name: source-revision                   # (!) required
tasks:
  - name: test
    params:
      - name: source-url
        value: $(params.source-url)
      - name: source-revision
        value: $(params.source-revision)
    taskSpec:
      params:
        - name: source-url
        - name: source-revision
      steps:
        - name: test
          image: gradle
          script: |-
            cd `mktemp -d`
            wget -qO- $(params.source-url) | tar xvz -m
            ./mvnw test
```

At this point, changes to the developer-provided Tekton Pipeline do not automatically trigger a re-run of the pipeline. That is, a new Tekton PipelineRun is not automatically created if a field in the Pipeline object is changed. As a workaround, the latest PipelineRun created can be deleted, which triggers a re-run.

Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script, as shown in the following example:

  ```
  apiVersion: tekton.dev/v1beta1
  kind: Pipeline
  metadata:
    name: developer-defined-tekton-pipeline
    labels:
      apps.tanzu.vmware.com/pipeline: test
  spec:
    #...
          steps:
            - name: test
              image: <image_that_has_JDK_and_Go>
              script: |-
                cd `mktemp -d`
                wget -qO- $(params.source-url) | tar xvz -m
                make test
  ```

- Update the pipeline resources to include labels that differentiate between the pipelines. For example, differentiate between Java and Go pipelines by adding labels for Java and Go:

  ```
  apiVersion: tekton.dev/v1beta1
  kind: Pipeline
  metadata:
    name: java-tests
    labels:
      apps.tanzu.vmware.com/pipeline: test
      apps.tanzu.vmware.com/language: java
  spec:
    #...
  ```

```
        steps:
          - name: test
            image: gradle
            script: |-
              # ...
              ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: go
spec:
  #...
        steps:
          - name: test
            image: golang
            script: |-
              # ...
              go test -v ./...
```

To match the correct pipeline, you add a `testing_pipeline_matching_labels` parameter to the workload. For example, if you want to match to the Java pipeline, you have the following `workload.yaml`:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: sample-java-app
  labels:
    apps.tanzu.vmware.com/has-tests: true
    apps.tanzu.vmware.com/workload-type: web
    app.kubernetes.io/part-of: sample-java-app
spec:
  params:
    - name: testing_pipeline_matching_labels
      value:
        apps.tanzu.vmware.com/pipeline: test
        apps.tanzu.vmware.com/language: java
  ...
```

This matches the workload to the pipeline with the `apps.tanzu.vmware.com/language: java` label.

Developer Workload

With the Tekton Pipeline object submitted to the same namespace as the one where the Workload will be submitted to, you can submit your Workload.

Regardless of the workflow being targeted (local development or gitops), the Workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload with tests enabled by means of the `has-tests` label.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    git:
     14 + |      ref:
     15 + |        branch: main
     16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     17 + |    subPath: tanzu-java-web-app
```

Install Out of the Box Supply Chain with Testing for Supply Chain Choreographer

This topic describes how you can install Out of the Box Supply Chain with Testing for Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Supply Chain with Testing. For more information about profiles, see Components and installation profiles.

The Out of the Box Supply Chain with Testing package provides a ClusterSupplyChain that brings an application from source code to a deployed instance that:

- Runs in a Kubernetes environment.
- Runs developer-provided tests in the form of Tekton/Pipeline objects to validate the source code before building container images.

Prerequisites

Before installing Out of the Box Supply Chain with Testing:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cartographer. For more information, see Install Supply Chain Choreographer.
- Install Out of the Box Delivery Basic
- Install Out of the Box Templates

Install

Install by following these steps:

1. Ensure you do not have Out of the Box Supply Chain With Testing and Scanning (`ootb-supply-chain-testing-scanning.tanzu.vmware.com`) installed:
    1. Run the following command:

```
tanzu package installed list --namespace tap-install
```

2. Verify `ootb-supply-chain-testing-scanning` is in the output:

```
NAME                                PACKAGE-NAME
ootb-delivery-basic                 ootb-delivery-basic.tanzu.vmware.com
ootb-supply-chain-basic             ootb-supply-chain-basic.tanzu.vmware.
com
ootb-templates                      ootb-templates.tanzu.vmware.com
```

3. If you see `ootb-supply-chain-testing-scanning` in the list, uninstall it by running:

```
tanzu package installed delete ootb-supply-chain-testing-scanning --names
pace tap-install
```

Example output:

```
Deleting installed package 'ootb-supply-chain-testing-scanning' in namesp
ace 'tap-install'.
Are you sure? [y/N]: y

| Uninstalling package 'ootb-supply-chain-testing-scanning' from namespac
e 'tap-install'
\ Getting package install for 'ootb-supply-chain-testing-scanning'
- Deleting package install 'ootb-supply-chain-testing-scanning' from name
space 'tap-install'
| Deleting admin role 'ootb-supply-chain-testing-scanning-tap-install-clu
ster-role'
| Deleting role binding 'ootb-supply-chain-testing-scanning-tap-install-c
luster-rolebinding'
| Deleting secret 'ootb-supply-chain-testing-scanning-tap-install-values'
| Deleting service account 'ootb-supply-chain-testing-scanning-tap-instal
l-sa'

 Uninstalled package 'ootb-supply-chain-testing-scanning' from namespace
'tap-install'
```

2. Verify that the values of the package can be configured by referencing the values below:

```
KEY                                 DESCRIPTION

registry.repository                  Name of the repository in the image regi
stry server where the application
                                     images from the workload should be pushe
d (required).

registry.server                      Name of the registry server where applic
ation images should be pushed to
                                     (required).

git_implementation                   Determines which git client library to u
se. Valid options are go-git or
                                     libgit2.

gitops.server_address                Default server address to be used for fo
rming Git URLs for pushing
                                     Kubernetes configuration produced by the
supply chain. This must
                                     include the scheme/protocol (e.g. http
s:// or ssh://)

gitops.repository_owner              Default project or user of the repositor
y. Used to create URLs for pushing
```

```
                                        Kubernetes configuration produced by the
supply chain.

gitops.repository_name                  Default repository name used for forming
Git URLs for pushing Kubernetes
                                        configuration produced by the supply cha
in.

gitops.username                         Default user name to be used for the com
mits produced by the supply chain.

gitops.branch                           Default branch to use for pushing Kubern
etes configuration files produced
                                        by the supply chain.

gitops.commit_message                   Default git commit message to write when
publishing Kubernetes
                                        configuration files produces by the supp
ly chain to git.

gitops.email                            Default user email to be used for the co
mmits produced by the supply chain.

gitops.ssh_secret                       Name of the default Secret containing SS
H credentials to lookup in the
                                        developer namespace for the supply chain
to fetch source code from and
                                        push configuration to.

gitops.commit_strategy                  Specification of how commits are made to
the branch; directly or through a
                                        pull request.

gitops.repository_prefix                DEPRECATED: Use server_address and repos
itory_owner instead.
                                        Default prefix to be used for forming Gi
t SSH URLs for pushing Kubernetes
                                        configuration produced by the supply cha
in.

gitops.pull_request.server_kind         The git source control platform used

gitops.pull_request.commit_branch       The branch to which commits will be mad
e, before opening a pull request
                                        to the branch specified in .gitops.branc
h If the string "" is specified,
                                        an essentially random string will be use
d for the branch name, in order
                                        to prevent collisions.

gitops.pull_request.pull_request_title  The title for the pull request

gitops.pull_request.pull_request_body   Any further information to add to the p
ull request

cluster_builder           Name of the Tanzu Build Service ClusterBuilder to
                          use by default on image objects managed by the supply
chain.

service_account           Name of the service account in the namespace where th
e Workload
                          is submitted to utilize for providing registry creden
tials to
                          Tanzu Build Service Image objects as well as deployin
g the
                          application.
```

3. Create a file named `ootb-supply-chain-testing-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

```
registry:
  server: REGISTRY-SERVER
  repository: REGISTRY-REPOSITORY

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh
  commit_strategy: direct

cluster_builder: default
service_account: default
```

Important

it's **required** that the `gitops.repository_prefix` field ends with a `/`.

4. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-testing \
  --package ootb-supply-chain-testing.tanzu.vmware.com \
  --version 0.7.0 \
  --namespace tap-install \
  --values-file ootb-supply-chain-testing-values.yaml
```

Example output:

```
\ Installing package 'ootb-supply-chain-testing.tanzu.vmware.com'
| Getting package metadata for 'ootb-supply-chain-testing.tanzu.vmware.com'
| Creating service account 'ootb-supply-chain-testing-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-testing-tap-install-cluster-ro
le'
| Creating cluster role binding 'ootb-supply-chain-testing-tap-install-cluster-
rolebinding'
| Creating secret 'ootb-supply-chain-testing-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'ootb-supply-chain-testing'
\ 'PackageInstall' resource install status: Reconciling

Added installed package 'ootb-supply-chain-testing' in namespace 'tap-install'
```

Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer.

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains supply chains that pass the source code through testing and vulnerability scanning, and also the container image.

This package includes all the capabilities of the Out of the Box Supply Chain With Testing, but adds source and image scanning using Grype.

Workloads that use source code or prebuilt images perform the following:

- Building from source code:
    1. Watching a Git Repository or local directory for changes
    2. Running tests from a developer-provided Tekton pipeline
    3. Scanning the source code for known vulnerabilities using Grype
    4. Building a container image out of the source code with Buildpacks
    5. Scanning the image for known vulnerabilities
    6. Applying operator-defined conventions to the container definition
    7. Deploying the application to the same cluster
- Using a prebuilt application image:
    1. Scanning the image for known vulnerabilities
    2. Applying operator-defined conventions to the container definition
    3. Creating a deliverable object for deploying the application to a cluster

Prerequisites

To use this supply chain, verify that:

- Tanzu Application Platform GUI is configured to enable CVE scan results. This configuration enables the Supply Chain Choreographer Tanzu Application Platform GUI plug-in to retrieve metadata about project packages and their vulnerabilities.
- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing **is NOT installed**.
- Out of the Box Supply Chain With Testing and Scanning **is installed**.
- Developer namespace is configured with the objects according to Out of the Box Supply Chain With Testing guidance. This supply chain is in addition to the Supply Chain with testing.
- (Optionally) install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

Verify that you have the supply chains with scanning, not with testing, installed. Run:

```
tanzu apps cluster-supply-chain list
```

```
NAME                       LABEL SELECTOR
source-test-scan-to-url    apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/w
orkload-type=web
source-to-url              apps.tanzu.vmware.com/workload-type=web
```

If you see `source-test-to-url` in the list, the setup is wrong. You **must not have the *source-test-to-url* installed** at the same time as *source-test-scan-to-url*.

Developer namespace

This example builds on the previous Out of the Box Supply Chain examples, so only additions are included here.

To ensure that you configured the namespace correctly, it is important that the namespace has the objects that you configured in the other supply chain setups:

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

  For more information about the preceding objects, see Out of the Box Supply Chain Basic.
- **Tekton pipeline**: A pipeline runs whenever the supply chain hits the stage of testing the source code.

  For more information, see Out of the Box Supply Chain Testing.

And the new objects, that you create here:

- **scan policy**: Defines what to do with the results taken from scanning the source code and image produced. For more information, see ScanPolicy section.
- **source scan template**: A template of how TaskRuns are created for scanning the source code. See ScanTemplate section.
- **image scan template**: A template of how TaskRuns are created for scanning the image produced by the supply chain. See ScanTemplate section.

The following section includes details about the new objects, compared to Out of the Box Supply Chain With Testing.

Updates to the developer namespace

For source and image scans, scan templates and scan policies must exist in the same namespace as the workload. These define:

- `ScanTemplate`: how to run a scan, allowing one to change details about the execution of the scan (either for images or source code)
- `ScanPolicy`: how to evaluate whether the artifacts scanned are compliant. For example, allowing one to be either very strict, or restrictive about particular vulnerabilities found.

The names of the objects **must** match the names in the example with default installation configurations. This is overriden either by using the `ootb_supply_chain_testing_scanning` package configuration in the `tap-values.yaml` file or by using workload parameters:

- To override by using the `ootb_supply_chain_testing_scanning` package configuration, make the following modification to your `tap-values.yaml` file and perform a Tanzu Application Platform update.

  ```
  ootb_supply_chain_testing_scanning:
    scanning:
      source:
        policy: SCAN-POLICY
        template: SCAN-TEMPLATE
      image:
        policy: SCAN-POLICY
        template: SCAN-TEMPLATE
  ```

  Where `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.

- To override through workload parameters, use the following commands. For more information, see Tanzu apps workload apply.

  ```
  tanzu apps workload apply WORKLOAD --param "scanning_source_policy=SCAN-POLICY"
  -n DEV-NAMESPACE
  tanzu apps workload apply WORKLOAD --param "scanning_source_template=SCAN-TEMPL
  ATE" -n DEV-NAMESPACE
  ```

  Where:

  - `WORKLOAD` is the name of the workload.
  - `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.
  - `DEV-NAMESPACE` is the developer namespace.

ScanPolicy

The ScanPolicy defines a set of rules to evaluate for a particular scan to consider the artifacts (image or source code) either compliant or not.

When a ImageScan or SourceScan is created to run a scan, those reference a policy whose name **must** match the following sample `scan-policy`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
```

```
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

See Writing Policy Templates.

ScanTemplate

A ScanTemplate defines the PodTemplateSpec used by a TaskRun to run a particular scan (image or source). When the supply chain initiates an ImageScan or SourceScan, they reference these templates which must live in the same namespace as the workload with the names matching the following:

- source scanning (`blob-source-scan-template`)
- image scanning (`private-image-scan-template`)

If you are targeting a namespace that does not match the one configured in the Tanzu Application Platform profiles, for example, if `grype.namespace` is not the same as the one you are writing the workload to, you can install these in such namespace by making use of the `tanzu package install` command as described in Install Supply Chain Security Tools - Scan:

1. Create a file named `ootb-supply-chain-basic-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

   ```
   grype:
     namespace: YOUR-DEV-NAMESPACE
     targetImagePullSecret: registry-credentials
   ```

2. With the configuration ready, install the templates by running:

   ```
   tanzu package install grype-scanner \
     --package grype.scanning.apps.tanzu.vmware.com \
     --version 1.0.0 \
     --namespace YOUR-DEV-NAMESPACE
   ```

Note

Although you can customize the templates, if you are following the Getting Started guide, VMware recommends that you follow what is provided in the installation of `grype.scanning.apps.tanzu.vmware.com`. This is created in the same namespace as configured by using `grype.namespace` in either Tanzu Application Platform profiles or individual component installation as in the earlier example. For more information, see About Source and Image Scans.

Enable storing scan results

To enable SCST - Scan to store scan results by using SCST - Store, see Developer namespace setup for exporting the SCST - Store CA certificate and authentication token to the developer namespace.

Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script. For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test
spec:
  #...
        steps:
          - name: test
            image: <image_that_has_JDK_and_Go>
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              make test
```

- Update the template to include labels that differentiate the pipelines. Then configure the labels to differentiate between pipelines. For example:

```
  selector:
     resource:
       apiVersion: tekton.dev/v1beta1
       kind: Pipeline
     matchingLabels:
       apps.tanzu.vmware.com/pipeline: test
+         apps.tanzu.vmware.com/language: #@ data.values.workload.metadata.labe
ls["apps.tanzu.vmware.com/language"]
```

The following example shows one namespace per-language pipeline:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: java-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: java
spec:
  #...
        steps:
          - name: test
            image: gradle
            script: |-
              # ...
              ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: go
spec:
  #...
        steps:
          - name: test
            image: golang
```

```
            script: |-
              # ...
              go test -v ./...
```

Developer workload

With the ScanPolicy and ScanTemplate objects, with the required names set, submitted to the same namespace where the workload are submitted, you are ready to submit your workload.

Regardless of the workflow being targeted, such as local development or gitops, the workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload as having tests enabled.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    git:
     14 + |      ref:
     15 + |        branch: main
     16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     17 + |    subPath: tanzu-java-web-app
```

CVE triage workflow

The Supply Chain halts progression if either a SourceScan (`sourcescans.scanning.apps.tanzu.vmware.com`) or an ImageScan (`imagescans.scanning.apps.tanzu.vmware.com`) fails policy enforcement through the ScanPolicy (`scanpolicies.scanning.apps.tanzu.vmware.com`). This can prevent source code from building or images deploying that contain vulnerabilities that are in violation of the user-defined scan policy. For information about learning how to handle these vulnerabilities and unblock your Supply Chain, see Triaging and Remediating CVEs.

Scan Images using a different scanner

Supply Chain Security Tools - Scan includes additional integrations for running an image scan using Snyk and VMware Carbon Black.

Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer.

This package contains Cartographer Supply Chains that tie together a series of Kubernetes resources that drive a developer-provided workload from source code to a Kubernetes configuration ready to be deployed to a cluster. It contains supply chains that pass the source code through testing and vulnerability scanning, and also the container image.

This package includes all the capabilities of the Out of the Box Supply Chain With Testing, but adds source and image scanning using Grype.

Workloads that use source code or prebuilt images perform the following:

- Building from source code:
    1. Watching a Git Repository or local directory for changes
    2. Running tests from a developer-provided Tekton pipeline
    3. Scanning the source code for known vulnerabilities using Grype
    4. Building a container image out of the source code with Buildpacks
    5. Scanning the image for known vulnerabilities
    6. Applying operator-defined conventions to the container definition
    7. Deploying the application to the same cluster
- Using a prebuilt application image:
    1. Scanning the image for known vulnerabilities
    2. Applying operator-defined conventions to the container definition
    3. Creating a deliverable object for deploying the application to a cluster

Prerequisites

To use this supply chain, verify that:

- Tanzu Application Platform GUI is configured to enable CVE scan results. This configuration enables the Supply Chain Choreographer Tanzu Application Platform GUI plug-in to retrieve metadata about project packages and their vulnerabilities.
- Out of the Box Templates is installed.
- Out of the Box Supply Chain With Testing **is NOT installed**.
- Out of the Box Supply Chain With Testing and Scanning **is installed**.
- Developer namespace is configured with the objects according to Out of the Box Supply Chain With Testing guidance. This supply chain is in addition to the Supply Chain with testing.
- (Optionally) install Out of the Box Delivery Basic, if you are willing to deploy the application to the same cluster as the workload and supply chains.

Verify that you have the supply chains with scanning, not with testing, installed. Run:

```
tanzu apps cluster-supply-chain list
```

```
NAME                        LABEL SELECTOR
source-test-scan-to-url     apps.tanzu.vmware.com/has-tests=true,apps.tanzu.vmware.com/w
orkload-type=web
source-to-url               apps.tanzu.vmware.com/workload-type=web
```

If you see `source-test-to-url` in the list, the setup is wrong. You **must not have the *source-test-to-url* installed** at the same time as *source-test-scan-to-url*.

Developer namespace

This example builds on the previous Out of the Box Supply Chain examples, so only additions are included here.

To ensure that you configured the namespace correctly, it is important that the namespace has the objects that you configured in the other supply chain setups:

- **registries secrets**: Kubernetes secrets of type `kubernetes.io/dockerconfigjson` that contain credentials for pushing and pulling the container images built by the supply chain and the installation of Tanzu Application Platform.
- **service account**: The identity to be used for any interaction with the Kubernetes API made by the supply chain.
- **rolebinding**: Grant to the identity the necessary roles for creating the resources prescribed by the supply chain.

  For more information about the preceding objects, see Out of the Box Supply Chain Basic.
- **Tekton pipeline**: A pipeline runs whenever the supply chain hits the stage of testing the source code.

  For more information, see Out of the Box Supply Chain Testing.

And the new objects, that you create here:

- **scan policy**: Defines what to do with the results taken from scanning the source code and image produced. For more information, see ScanPolicy section.
- **source scan template**: A template of how TaskRuns are created for scanning the source code. See ScanTemplate section.
- **image scan template**: A template of how TaskRuns are created for scanning the image produced by the supply chain. See ScanTemplate section.

The following section includes details about the new objects, compared to Out of the Box Supply Chain With Testing.

Updates to the developer namespace

For source and image scans, scan templates and scan policies must exist in the same namespace as the workload. These define:

- `ScanTemplate`: how to run a scan, allowing one to change details about the execution of the scan (either for images or source code)
- `ScanPolicy`: how to evaluate whether the artifacts scanned are compliant. For example, allowing one to be either very strict, or restrictive about particular vulnerabilities found.

The names of the objects **must** match the names in the example with default installation configurations. This is overriden either by using the `ootb_supply_chain_testing_scanning` package configuration in the `tap-values.yaml` file or by using workload parameters:

- To override by using the `ootb_supply_chain_testing_scanning` package configuration, make the following modification to your `tap-values.yaml` file and perform a Tanzu Application Platform update.

  ```
  ootb_supply_chain_testing_scanning:
    scanning:
      source:
        policy: SCAN-POLICY
        template: SCAN-TEMPLATE
      image:
        policy: SCAN-POLICY
        template: SCAN-TEMPLATE
  ```

  Where `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.

- To override through workload parameters, use the following commands. For more information, see Tanzu apps workload apply.

  ```
  tanzu apps workload apply WORKLOAD --param "scanning_source_policy=SCAN-POLICY"
  -n DEV-NAMESPACE
  tanzu apps workload apply WORKLOAD --param "scanning_source_template=SCAN-TEMPL
  ATE" -n DEV-NAMESPACE
  ```

  Where:

  - `WORKLOAD` is the name of the workload.
  - `SCAN-POLICY` and `SCAN-TEMPLATE` are the names of the `ScanPolicy` and `ScanTemplate`.
  - `DEV-NAMESPACE` is the developer namespace.

ScanPolicy

The ScanPolicy defines a set of rules to evaluate for a particular scan to consider the artifacts (image or source code) either compliant or not.

When a ImageScan or SourceScan is created to run a scan, those reference a policy whose name **must** match the following sample `scan-policy`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
```

```
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

See Writing Policy Templates.

ScanTemplate

A ScanTemplate defines the PodTemplateSpec used by a TaskRun to run a particular scan (image or source). When the supply chain initiates an ImageScan or SourceScan, they reference these templates which must live in the same namespace as the workload with the names matching the following:

- source scanning (`blob-source-scan-template`)
- image scanning (`private-image-scan-template`)

If you are targeting a namespace that does not match the one configured in the Tanzu Application Platform profiles, for example, if `grype.namespace` is not the same as the one you are writing the workload to, you can install these in such namespace by making use of the `tanzu package install` command as described in Install Supply Chain Security Tools - Scan:

1. Create a file named `ootb-supply-chain-basic-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

   ```
   grype:
     namespace: YOUR-DEV-NAMESPACE
     targetImagePullSecret: registry-credentials
   ```

2. With the configuration ready, install the templates by running:

   ```
   tanzu package install grype-scanner \
     --package grype.scanning.apps.tanzu.vmware.com \
     --version 1.0.0 \
     --namespace YOUR-DEV-NAMESPACE
   ```

Note

Although you can customize the templates, if you are following the Getting Started guide, VMware recommends that you follow what is provided in the installation of `grype.scanning.apps.tanzu.vmware.com`. This is created in the same namespace as configured by using `grype.namespace` in either Tanzu Application Platform profiles or

individual component installation as in the earlier example. For more information, see About Source and Image Scans.

Enable storing scan results

To enable SCST - Scan to store scan results by using SCST - Store, see Developer namespace setup for exporting the SCST - Store CA certificate and authentication token to the developer namespace.

Allow multiple Tekton pipelines in a namespace

You can configure your developer namespace to include more than one pipeline using either of the following methods:

- Use a single pipeline running on a container image that includes testing tools and runs a common script to execute tests. This allows you to accommodate multiple workloads based in different languages in the same namespace that use a common make test script. For example:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-tekton-pipeline
  labels:
    apps.tanzu.vmware.com/pipeline: test
spec:
  #...
        steps:
          - name: test
            image: <image_that_has_JDK_and_Go>
            script: |-
              cd `mktemp -d`
              wget -qO- $(params.source-url) | tar xvz -m
              make test
```

- Update the template to include labels that differentiate the pipelines. Then configure the labels to differentiate between pipelines. For example:

```
  selector:
     resource:
       apiVersion: tekton.dev/v1beta1
       kind: Pipeline
     matchingLabels:
       apps.tanzu.vmware.com/pipeline: test
+         apps.tanzu.vmware.com/language: #@ data.values.workload.metadata.labe
ls["apps.tanzu.vmware.com/language"]
```

  The following example shows one namespace per-language pipeline:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: java-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: java
spec:
  #...
        steps:
```

```
          - name: test
            image: gradle
            script: |-
              # ...
              ./mvnw test
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: go-tests
  labels:
    apps.tanzu.vmware.com/pipeline: test
    apps.tanzu.vmware.com/language: go
spec:
  #...
        steps:
          - name: test
            image: golang
            script: |-
              # ...
              go test -v ./...
```

Developer workload

With the ScanPolicy and ScanTemplate objects, with the required names set, submitted to the same namespace where the workload are submitted, you are ready to submit your workload.

Regardless of the workflow being targeted, such as local development or gitops, the workload configuration details are the same as in Out of the Box Supply Chain Basic, except that you mark the workload as having tests enabled.

For example:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label apps.tanzu.vmware.com/has-tests=true \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      9 + |  name: tanzu-java-web-app
     10 + |  namespace: default
     11 + |spec:
     12 + |  source:
     13 + |    git:
     14 + |      ref:
     15 + |        branch: main
     16 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     17 + |    subPath: tanzu-java-web-app
```

CVE triage workflow

The Supply Chain halts progression if either a SourceScan (`sourcescans.scanning.apps.tanzu.vmware.com`) or an ImageScan (`imagescans.scanning.apps.tanzu.vmware.com`) fails policy enforcement through the ScanPolicy (`scanpolicies.scanning.apps.tanzu.vmware.com`). This can prevent source code from building or images deploying that contain vulnerabilities that are in violation of the user-defined scan policy. For information about learning how to handle these vulnerabilities and unblock your Supply Chain, see Triaging and Remediating CVEs.

Scan Images using a different scanner

Supply Chain Security Tools - Scan includes additional integrations for running an image scan using Snyk and VMware Carbon Black.

Install Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer

This topic describes how you can install Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Supply Chain with Testing and Scanning. For more information about profiles, see Components and installation profiles.

The Out of the Box Supply Chain with Testing and Scanning package provides a ClusterSupplyChain that brings an application from source code to a deployed instance that:

- Runs in a Kubernetes environment.
- Performs validations in terms of running application tests.
- Scans the source code and image for vulnerabilities.

Prerequisites

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cartographer. For more information, see Install Supply Chain Choreographer.
- Install Out of the Box Delivery Basic
- Install Out of the Box Templates

Install

To install Out of the Box Supply Chain with Testing and Scanning:

1. Ensure you do not have Out of The Box Supply Chain With Testing (`ootb-supply-chain-testing.tanzu.vmware.com`) installed:
    1. Run the following command:

        ```
        tanzu package installed list --namespace tap-install
        ```

    2. Verify `ootb-supply-chain-testing` is in the output:

```
NAME                                PACKAGE-NAME
ootb-delivery-basic                 ootb-delivery-basic.tanzu.vmware.com
ootb-supply-chain-basic             ootb-supply-chain-basic.tanzu.vmware.
com
ootb-templates                      ootb-templates.tanzu.vmware.com
```

3. If you see `ootb-supply-chain-testing` in the list, uninstall it by running:

```
tanzu package installed delete ootb-supply-chain-testing --namespace tap-
install
```

Example output:

```
Deleting installed package 'ootb-supply-chain-testing' in namespace 'tap-
install'.
Are you sure? [y/N]: y

| Uninstalling package 'ootb-supply-chain-testing' from namespace 'tap-in
stall'
\ Getting package install for 'ootb-supply-chain-testing'
- Deleting package install 'ootb-supply-chain-testing' from namespace 'ta
p-install'
| Deleting admin role 'ootb-supply-chain-testing-tap-install-cluster-rol
e'
| Deleting role binding 'ootb-supply-chain-testing-tap-install-cluster-ro
lebinding'
| Deleting secret 'ootb-supply-chain-testing-tap-install-values'
| Deleting service account 'ootb-supply-chain-testing-tap-install-sa'

 Uninstalled package 'ootb-supply-chain-testing' from namespace 'tap-inst
all'
```

2. Check the values of the package that can be configured by running:

```
tanzu package available get ootb-supply-chain-testing-scanning.tanzu.vmware.co
m/0.7.0 \
  --values-schema \
  -n tap-install
```

For example:

```
KEY                                 DESCRIPTION

registry.repository                  Name of the repository in the image regi
stry server where the application
                                     images from the workload should be pushe
d (required).

registry.server                      Name of the registry server where applic
ation images should be pushed to
                                     (required).

git_implementation                   Determines which git client library to u
se. Valid options are go-git or
                                     libgit2.

gitops.server_address                Default server address to be used for fo
rming Git URLs for pushing
                                     Kubernetes configuration produced by the
supply chain. This must
                                     include the scheme/protocol (e.g. http
s:// or ssh://)
```

```
gitops.repository_owner                Default project or user of the repositor
y. Used to create URLs for pushing
                                       Kubernetes configuration produced by the
supply chain.

gitops.repository_name                 Default repository name used for forming
Git URLs for pushing Kubernetes
                                       configuration produced by the supply cha
in.

gitops.username                        Default user name to be used for the com
mits produced by the supply chain.

gitops.branch                          Default branch to use for pushing Kubern
etes configuration files produced
                                       by the supply chain.

gitops.commit_message                  Default git commit message to write when
publishing Kubernetes
                                       configuration files produces by the supp
ly chain to git.

gitops.email                           Default user email to be used for the co
mmits produced by the supply chain.

gitops.ssh_secret                      Name of the default Secret containing SS
H credentials to lookup in the
                                       developer namespace for the supply chain
to fetch source code from and
                                       push configuration to.

gitops.commit_strategy                 Specification of how commits are made to
the branch; directly or through a
                                       pull request.

gitops.repository_prefix               DEPRECATED: Use server_address and repos
itory_owner instead.
                                       Default prefix to be used for forming Gi
t SSH URLs for pushing Kubernetes
                                       configuration produced by the supply cha
in.

gitops.pull_request.server_kind         The git source control platform used

gitops.pull_request.commit_branch       The branch to which commits will be mad
e, before opening a pull request
                                       to the branch specified in .gitops.branc
h If the string "" is specified,
                                       an essentially random string will be use
d for the branch name, in order
                                       to prevent collisions.

gitops.pull_request.pull_request_title  The title for the pull request

gitops.pull_request.pull_request_body   Any further information to add to the p
ull request

cluster_builder           Name of the Tanzu Build Service ClusterBuilder to
                          use by default on image objects managed by the supply
chain.

service_account           Name of the service account in the namespace where th
e Workload
                          is submitted to utilize for providing registry creden
tials to
                          Tanzu Build Service Image objects as well as deployin
```

```
g the
                            application.
```

3. Create a file named `ootb-supply-chain-testing-scanning-values.yaml` that specifies the corresponding values to the properties you want to change. For example:

```
registry:
  server: REGISTRY-SERVER
  repository: REGISTRY-REPOSITORY

gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  branch: main
  username: supplychain
  email: supplychain
  commit_message: supplychain@cluster.local
  ssh_secret: git-ssh
  commit_strategy: direct

cluster_builder: default
service_account: default
```

Important

The `gitops.repository_prefix` field must end with `/`.

4. With the configuration ready, install the package by running:

```
tanzu package install ootb-supply-chain-testing-scanning \
  --package ootb-supply-chain-testing-scanning.tanzu.vmware.com \
  --version 0.7.0 \
  --namespace tap-install \
  --values-file ootb-supply-chain-testing-scanning-values.yaml
```

Example output:

```
\ Installing package 'ootb-supply-chain-testing-scanning.tanzu.vmware.com'
| Getting package metadata for 'ootb-supply-chain-testing-scanning.tanzu.vmwar
e.com'
| Creating service account 'ootb-supply-chain-testing-scanning-tap-install-sa'
| Creating cluster admin role 'ootb-supply-chain-testing-scanning-tap-install-c
luster-role'
| Creating cluster role binding 'ootb-supply-chain-testing-scanning-tap-install
-cluster-rolebinding'
| Creating secret 'ootb-supply-chain-testing-scanning-tap-install-values'
| Creating package resource
- Waiting for 'PackageInstall' reconciliation for 'ootb-supply-chain-testing-sc
anning'
\ 'PackageInstall' resource install status: Reconciling

Added installed package 'ootb-supply-chain-testing-scanning' in namespace 'tap-
install'
```

Out of the Box Templates for Supply Chain Choreographer

This topic describes the templates you can use with Supply Chain Choreographer.

Templates define Kubernetes objects based on configuration in the workload, supply chain Tanzu Application Platform values, and results output from other templated objects. A supply chain

organizes a set of templates into a directed acyclic graph. This package contains templates that are used by the Out of the Box Supply Chains and the Out of the Box Delivery. You must install this package to have Workloads delivered properly.

The OOTB Template package includes:

- Cartographer Templates: See reference
- Cartographer ClusterRunTemplates: See reference
- Tekton ClusterTasks
- ClusterRoles
- openshift SecurityContextConstraints

For information about OOTB Supply Chains and Delivery, see:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain with Testing
- Out of the Box Supply Chain with Testing and Scanning
- Out of the Box Delivery Basic

Out of the Box Templates for Supply Chain Choreographer

This topic describes the templates you can use with Supply Chain Choreographer.

Templates define Kubernetes objects based on configuration in the workload, supply chain Tanzu Application Platform values, and results output from other templated objects. A supply chain organizes a set of templates into a directed acyclic graph. This package contains templates that are used by the Out of the Box Supply Chains and the Out of the Box Delivery. You must install this package to have Workloads delivered properly.

The OOTB Template package includes:

- Cartographer Templates: See reference
- Cartographer ClusterRunTemplates: See reference
- Tekton ClusterTasks
- ClusterRoles
- openshift SecurityContextConstraints

For information about OOTB Supply Chains and Delivery, see:

- Out of the Box Supply Chain Basic
- Out of the Box Supply Chain with Testing
- Out of the Box Supply Chain with Testing and Scanning
- Out of the Box Delivery Basic

Install Out of the Box Templates

This document describes how to install Out of the Box Templates from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Templates. For more information about profiles, see Components and

installation profiles.

The Out of the Box Templates package is used by all the Out of the Box Supply Chains to provide the templates that are used by the Supply Chains to create the objects that drive source code all the way to a deployed application in a cluster.

Prerequisites

Before installing Out of the Box Templates:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cartographer. For more information, see Install Supply Chain Choreographer.
- Install Tekton Pipelines.

Install

To install Out of the Box Templates:

1. View the configurable values of the package by running:

   ```
   tanzu package available get ootb-templates.tanzu.vmware.com/0.7.0 \
     --values-schema \
     -n tap-install
   ```

   For example:

   ```
   KEY                  DEFAULT  TYPE    DESCRIPTION
   excluded_templates   []       array   List of templates to exclude from the
                                         installation (e.g. ['git-writer'])
   ```

2. Create a file named `ootb-templates.yaml` that specifies the corresponding values to the properties you want to change.

   For example, the contents of the file might look like this:

   ```
   excluded_templates: []
   ```

3. After the configuration is ready, install the package by running:

   ```
   tanzu package install ootb-templates \
     --package ootb-templates.tanzu.vmware.com \
     --version 0.7.0 \
     --namespace tap-install \
     --values-file ootb-templates-values.yaml
   ```

   Example output:

   ```
   \ Installing package 'ootb-templates.tanzu.vmware.com'
   | Getting package metadata for 'ootb-templates.tanzu.vmware.com'
   | Creating service account 'ootb-templates-tap-install-sa'
   | Creating cluster admin role 'ootb-templates-tap-install-cluster-role'
   | Creating cluster role binding 'ootb-templates-tap-install-cluster-rolebindin
   g'
   | Creating secret 'ootb-templates-tap-install-values'
   | Creating package resource
   - Waiting for 'PackageInstall' reconciliation for 'ootb-templates'
   / 'PackageInstall' resource install status: Reconciling
   ```

```
Added installed package 'ootb-templates' in namespace 'tap-install'
```

Out of the Box Delivery Basic for Supply Chain Choreographer

This topic is an overview of the Out of the Box Delivery Basic package for Supply Chain Choreographer.

This package provides a reusable ClusterDelivery object that delivers the Kubernetes configuration that the Out of the Box Supply Chain produces to an environment, including Basic, Testing, and Testing With Scanning supply chains.

Prerequisites

To make use of this package you must have installed:

- Supply Chain Cartographer
- Out of the Box Templates

Using Out of the Box Delivery Basic

Out of the Box Delivery Basic support both GitOps and local development workflows:

```
GITOPS

    Deliverable:
      points at a git repository where source code is found and
      kubernetes configuration is pushed to


LOCAL DEVELOPMENT

    Deliverable:

      points at a container image registry where the supplychain
      pushes source code and configuration to


---

DELIVERY

    takes a Deliverable (local or gitops) and passes is through
    a series of resources:


           config-provider  <---[config]--- deployer
                 .                             .
                 .                             .
    GitRepository/ImageRepository         kapp-ctrl/App
                                                - knative/Service
                                                - ResourceClaim
                                                - ServiceBinding
                                                ...
```

You must install this package to have Workloads delivered properly with the Basic, Testing, and Testing With Scanning Out of the Box Supply Chains.

Consumers do not interact directly with this package. Instead, this package is used after the supply chains create a carto.run/Deliverable object to express the intention of having the Workloads that go through them delivered to an environment. The environment is the same Kubernetes cluster as the Supply Chains.

More information

- Reference
- Installation

Out of the Box Delivery Basic for Supply Chain Choreographer

This topic is an overview of the Out of the Box Delivery Basic package for Supply Chain Choreographer.

This package provides a reusable ClusterDelivery object that delivers the Kubernetes configuration that the Out of the Box Supply Chain produces to an environment, including Basic, Testing, and Testing With Scanning supply chains.

Prerequisites

To make use of this package you must have installed:

- Supply Chain Cartographer
- Out of the Box Templates

Using Out of the Box Delivery Basic

Out of the Box Delivery Basic support both GitOps and local development workflows:

```
GITOPS

    Deliverable:
      points at a git repository where source code is found and
      kubernetes configuration is pushed to


LOCAL DEVELOPMENT

    Deliverable:

      points at a container image registry where the supplychain
      pushes source code and configuration to


---

DELIVERY

    takes a Deliverable (local or gitops) and passes is through
    a series of resources:


           config-provider  <---[config]--- deployer
                 .                             .
                 .                             .
    GitRepository/ImageRepository         kapp-ctrl/App
                                                - knative/Service
```

```
                                                - ResourceClaim
                                                - ServiceBinding
                                                ...
```

You must install this package to have Workloads delivered properly with the Basic, Testing, and Testing With Scanning Out of the Box Supply Chains.

Consumers do not interact directly with this package. Instead, this package is used after the supply chains create a carto.run/Deliverable object to express the intention of having the Workloads that go through them delivered to an environment. The environment is the same Kubernetes cluster as the Supply Chains.

More information

- Reference
- Installation

Install Out of the Box Delivery Basic for Supply Chain Choreographer

This topic shows you how to install the Out of the Box Delivery Basic package for Supply Chain Choreographer from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Out of the Box Delivery Basic. For more information about profiles, see Components and installation profiles.

The Out of the Box Delivery Basic package is used by all the Out of the Box Supply Chains to deliver the objects that have been produced by them to a Kubernetes environment.

Prerequisites

Before installing Out of the Box Delivery Basic:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cartographer. For more information, see Install Supply Chain Choreographer.

Install

To install Out of the Box Delivery Basic:

1. Familiarize yourself with the set of values of the package that can be configured by running:

   ```
   tanzu package available get ootb-delivery-basic.tanzu.vmware.com/0.7.0 \
     --values-schema \
     -n tap-install
   ```

   For example:

   ```
   KEY                 DEFAULT  TYPE    DESCRIPTION
   service_account     default  string  Name of the service account in the
                                        namespace where the Deliverable is
                                        submitted to.
   ```

```
git_implementation   go-git   string  Which git client library to use.
                                       Valid options are go-git or libgit2.
```

2. Create a file named `ootb-delivery-basic-values.yaml` that specifies the corresponding values to the properties you want to change.

   For example, the contents of the file might look like this:

   ```
   service_account: default
   ```

3. With the configuration ready, install the package by running:

   ```
   tanzu package install ootb-delivery-basic \
     --package ootb-delivery-basic.tanzu.vmware.com \
     --version 0.7.0 \
     --namespace tap-install \
     --values-file ootb-delivery-basic-values.yaml
   ```

   Example output:

   ```
   \ Installing package 'ootb-delivery-basic.tanzu.vmware.com'
   | Getting package metadata for 'ootb-delivery-basic.tanzu.vmware.com'
   | Creating service account 'ootb-delivery-basic-tap-install-sa'
   | Creating cluster admin role 'ootb-delivery-basic-tap-install-cluster-role'
   | Creating cluster role binding 'ootb-delivery-basic-tap-install-cluster-rolebi
   nding'
   | Creating secret 'ootb-delivery-basic-tap-install-values'
   | Creating package resource
   - Waiting for 'PackageInstall' reconciliation for 'ootb-delivery-basic'
   / 'PackageInstall' resource install status: Reconciling

    Added installed package 'ootb-delivery-basic' in namespace 'tap-install'
   ```

How-to guides for Supply Chain Choreographer for Tanzu

This topic describes the how-to guides you can use for Supply Chain Choreographer for Tanzu.

How-to guides

The following how-to guides apply to Supply Chain Choreographer for Tanzu:

- Install Supply Chain Choreographer
- Install Out of the Box Delivery Basic
- Install Out of the Box Supply Chain Basic
- Install Out of the Box Supply Chain with Testing
- Install Out of the Box Supply Chain with Testing and Scanning
- Install Out of the Box Templates
- Tanzu Build Service Integration
- Building from source
- Git authentication
- Output Carvel Packages from your Supply Chain
- Deploy Carvel Packages using Carvel App CR

- Deploy Carvel Packages using Flux CD Kustomization
- Use Blue-green deployments with Contour and Carvel Packages

Out of the Box Supply Chain with testing on Jenkins for Supply Chain Choreographer

This topic provides an overview of Out of the Box Supply Chain with testing on Jenkins for Supply Chain Choreographer.

The Out of the Box templates package now includes a Tekton `ClusterTask` resource, which triggers a build for a specified Jenkins job.

You can configure the Jenkins task in both the Out of the Box Supply Chain with Testing and Out of the Box Supply Chain With Testing and Scanning to trigger a Jenkins job. The task is implemented as a Tekton `ClusterTask` and can now run from a Tekton `Pipeline`.

Prerequisites

Follow the instructions from either Out of the Box Supply Chain With Testing or Out of the Box Supply Chain With Testing and Scanning to install the required packages. You need to set up only one of these packages.

Either of these Supply Chains is able to use the Jenkins service during the `source-tester` phase of the pipeline.

Using the Out of the Box Jenkins Task

The intent of the Jenkins task provided via Out of the Box templates is to help Tanzu Application Platform users to integrate with and make use of the modern application deployment pipeline provided by our platform while maintaining their existing test suites on their Jenkins services.

The Out of the Box Jenkins task makes use of an existing Jenkins Job to run test suites on source code.

1. Configuring a Jenkins job in an existing Jenkins Pipeline

This section of the guide shows how to configure a Jenkins job that can be kicked off by the Tanzu Application Platform Jenkins task.

Example Jenkins Job

Here is an example of a script that can be added to your pipeline that specifies source url and source revision information for your source code target. This example uses a Jenkins instance that is deployed on a Kubernetes cluster although this is not the only possible configuration for a Jenkins instance.

```
#!/bin/env groovy

pipeline {
  agent {
    // Use an agent that is appropriate
    // for your Jenkins installation.
    // This is only an example
    kubernetes {
      label 'maven'
    }
  }
```

```
  stages {
    stage('Checkout code') {
      steps {
        script {
          sourceUrl = params.SOURCE-REVISION
          indexSlash = sourceUrl.indexOf("/")
          revision = sourceUrl.substring(indexSlash + 1)
        }
        sh "git clone ${params.GIT-URL} target"
        dir("target") {
          sh "git checkout ${revision}"
        }
      }
    }

    stage('Maven test') {
      steps {
        container('maven') {
          dir("target") {
            // Example tests with maven
            sh "mvn clean test --no-transfer-progress"
          }
        }
      }
    }
  }
}
```

Where:

- `SOURCE_URL` **string** The URL of the source code being tested. The `source-provider` resource in the supply chain provides this code and is only resolvable inside the Kubernetes cluster. This URL is only useful if your Jenkins service is running inside the cluster or if there is ingress set up and the Jenkins service can make requests to services inside the cluster.
- `SOURCE_REVISION` **string** The revision of the source code being tested. The format of this value can vary depending on the implementation of the `source_provider` resource. If the `source-provider` is the Flux CD `GitRepository` resource, then the value of the `SOURCE_REVISION` is the Git branch name followed by the commit SHA, both separated by a (`/`) slash character. For example, `main/2b1ed6c3c4f74f15b0e4de2732234eafd050eb1ca`. Your Jenkins pipeline script must extract the commit SHA from the `SOURCE_REVISION` to be useful.

Note

If you can't use the `SOURCE_URL` because your Jenkins service cannot make requests into the Kubernetes cluster, you can supply the source code URL to the Jenkins job with other parameters instead.

The following fields will also be required in the Jenkins Job definition

- `SOURCE-REVISION` **string**
- `GIT-URL` **string**

To configure your `Workload` to pass the `GIT-URL` parameter into the Jenkins task:

```
tanzu apps workload create workload \
  --namespace your-test-namespace \
  --git-branch main \
  --git-repo https://your.git/repository.git \
  --label apps.tanzu.vmware.com/has-tests=true \
```

```
  --label app.kubernetes.io/part-of=test-workload \
  --param-yaml testing_pipeline_matching_labels='{"apps.tanzu.vmware.com/pipeline":"je
nkins-pipeline"}' \
  --param-yaml testing_pipeline_params='{"secret-name":"my-secret","job-name":"jenkins
-job-name","job-params":"[{\"name\":\"GIT_URL\",\"value\":\"https://your.git/repositor
y.git\"}]"}' \
  --type web \
  --yes
```

The `Workload` is described in the later Developer Workload section.

2. Create a secret with auth credentials

A secret must be created in the developer namespace to contain the credentials required to authenticate and interact with your Jenkins instance's builds. The following properties are required:

- `url` **required**: URL of the Jenkins instance that hosts the job, including the scheme. For example: `https://my-jenkins.com`.
- `username` **required**: User name of the user that has access to trigger a build on Jenkins.
- `password` **required**: Password of the user that has access to trigger a build on Jenkins.
- `ca-cert` **optional**: The PEM-encoded CA certificate to verify the Jenkins instance identity.

Use the Kubernetes CLI tool (kubectl) to create the above secret. You can provide the optional PEM-encoded CA certificate as a file using the `--from-file` flag as shown below:

```
kubectl create secret generic my-secret \
  --from-literal=url=https://jenkins.instance \
  --from-literal=username=literal-username \
  --from-file=password=/path/to/file/with/password.txt \
  --from-file=ca-cert=/path/to/ca-certificate.pem \
```

The expected format of the secret is will be as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: MY-SECRET # secret name that will be referenced by the workload
type: Opaque
stringData:
  url: JENKINS-URL # target jenkins instance url
  username: USERNAME # jenkins username
  password: PASSWORD # jenkins password
  ca-cert: PEM-CA-CERT # PEM encoded certificate
```

3. Create a Tekton pipeline

The developer must create a Tekton `Pipeline` object with the following parameters:

- `source-url`, **required**: An HTTP address where a `.tar.gz` file containing all the source code being tested is supplied.
- `source-revision`, **required**: The revision of the commit or image reference found by the `source-provider`.
- `secret-name`, **required**: The secret that contains the URL, user name, password, and certificate (optional) to the Jenkins instance that houses the job that is required to run.
- `job-name`, **required**: The name of the Jenkins job that is required to run.
- `job-params`, **required**: A list of key-value pairs, encoded as a JSON string, that passes in parameters needed for the Jenkins job.

Tasks:

- `jenkins-task`, **required**: This `ClusterTask` is one of the tasks that the pipeline runs to trigger the Jenkins job. It is installed in the cluster by the **Out of the Box Templates** package.

Results:

- `jenkins-job-url`: A string result that outputs the URL of the Jenkins build that the Tekton task triggered. The `jenkins-task ClusterTask` populates the output.

Here is an example of how to create a tekton pipeline with the required parameters

```
cat <<EOF | kubectl apply -f -
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: developer-defined-jenkins-tekton-pipeline
  namespace: developer-namespace
  labels:
    #! This label should be provided to the Workload so that
    #! the supply chain can find this pipeline
    apps.tanzu.vmware.com/pipeline: jenkins-pipeline
spec:
  results:
  - name: jenkins-job-url   #! To show the job URL on the
    #! Tanzu Application Platform GUI
    value: $(tasks.jenkins-task.results.jenkins-job-url)
  params:
  - name: source-url        #! Required
  - name: source-revision   #! Required
  - name: secret-name       #! Required
  - name: job-name          #! Required
  - name: job-params        #! Required
  tasks:
  #! Required: Include the built-in task that triggers the
  #! given job in Jenkins
  - name: jenkins-task
    taskRef:
      name: jenkins-task
      kind: ClusterTask
    params:
      - name: source-url
        value: $(params.source-url)
      - name: source-revision
        value: $(params.source-revision)
      - name: secret-name
        value: $(params.secret-name)
      - name: job-name
        value: $(params.job-name)
      - name: job-params
        value: $(params.job-params)
EOF
```

4. Patching the default Service Account

Tanzu Application Platform includes a Namespace Provisioner which is not enabled by default. This section of the guide assumes that the user is not using the Namespace Provisioner.

The `jenkins-task ClusterTask` resource uses a container image with the Jenkins Adapter application to trigger the Jenkins job and wait for it to complete. This container image is distributed with Tanzu Application Platform on VMware Tanzu Network, but it is not installed at the same time

as the other packages. It is pulled at the time that the supply chain executes the job. As a result, it does not implicitly have access to the `imagePullSecrets` with the required credentials.

Important

The `ServiceAccount` that a developer can configure with their `Workload` is *not* passed to the task and is not used to pull the Jenkins Adapter container image. If you followed the Tanzu Application Platform Install Guide, then you have a `Secret` named `tap-registry` in each of your cluster's namespaces. You can patch the default Service Account in your workload's namespace so that your supply chain can pull the Jenkins Adapter image. For example:

```
kubectl patch serviceaccount default \
  --patch '{"imagePullSecrets": [{"name": "tap-registry"}]}' \
  --namespace developer-namespace
```

5. Create a Developer Workload

Submit your `Workload` to the same namespace as the Tekton `Pipeline` defined earlier.

To enable the supply chain to run Jenkins tasks, the `Workload` must include the following parameters:

```
parameters:

  #! Required: selects the pipeline
  - name: testing_pipeline_matching_labels
    value:
      #! This label must match the label on the pipeline created earlier
      apps.tanzu.vmware.com/pipeline: jenkins-pipeline

  #! Required: Passes parameters to pipeline
  - name: testing_pipeline_params
    value:

      #! Required: Name of the Jenkins job
      job-name: my-jenkins-job

      #! Required: The secret created earlier to access Jenkins
      secret-name: my-secret

      #! Required: The `job-params` element is required, but the parameter string
      #! might be empty. If empty, then set this value to `[]`.  If non-empty then the
      #! value contains a JSON-encoded list of parameters to pass to the Jenkins job.
      #! Ensure that the quotation marks inside the JSON-encoded string are escaped.
      job-params: "[{\"name\":\"A\",\"value\":\"x\"},{\"name\":\"B\",\"value\":\"y
\"},...]"
```

You can create the workload by using the `apps` CLI plug-in as shown below:

```
readonly GIT_BRANCH="my-git-branch"
readonly WORKLOAD_NAME="my-workload-name"
readonly GITHUB_REPO="github-repository-url"
readonly DEVELOPER_WORKSPACE_NAME="my-developer-namespace"

tanzu apps workload create "${WORKLOAD_NAME}" \
  --namespace "${DEVELOPER_WORKSPACE_NAME}" \
  --git-branch "${GIT_BRANCH}" \
  --git-repo "${GITHUB_REPO}" \
  --label apps.tanzu.vmware.com/has-tests=true \
```

```
  --label app.kubernetes.io/part-of="${WORKLOAAD_NAME}" \
  --param-yaml testing_pipeline_matching_labels='{"apps.tanzu.vmware.com/pipeline":"je
nkins-pipeline"}' \
  --param-yaml testing_pipeline_params='{"secret-name":"jenkins-secret", "job-name":
"jenkins-job", "job-params":"[{"name":"GIT-URL", "value":"https://github.com/spring-pr
ojects/spring-petclinic"}, {"name":"GIT-BRANCH", "value":"main"}]"}'\
  --type web
```

Where:

- `GIT-URL` is the URL of your GitHub repository.
- `GIT-BRANCH` is the branch you want to target.

The value of the `job-params` parameter is a list of zero-or-more parameters that are sent to the Jenkins job. The parameter is entered into the `Workload` as a list of name-value pairs as shown in the example above.

Important

None of the fields in the `Workload` resource are implicitly passed to the Jenkins job. You have to set them in the `job-params` explicitly. An exception to this is the `SOURCE_URL` and `SOURCE_REVISION` parameters are sent to the Jenkins job implicitly by the Jenkins Adapter trigger application. For example, you can use the `SOURCE_REVISION` to verify which commit SHA to test. See Making a Jenkins Test Job earlier for details about how to use the Git URL and source revision in a Jenkins test job.

Watch the quoting of the `job-params` value closely. In the earlier `tanzu apps workload create` example, the `job-params` value is a string with a JSON structure in it. The value of the `--param-yaml testing_pipeline_params` parameter is a JSON string. Add backslash (`\`) escape characters before the double quote characters (`"`) in the `job-params` value.

Example output from the `tanzu apps workload create` command:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: my-workload-name
      7 + |    apps.tanzu.vmware.com/has-tests: "true"
      8 + |  name: my-workload-name
      9 + |  namespace: developer-namespace
     10 + |spec:
     11 + |  params:
     12 + |  - name: testing_pipeline_matching_labels
     13 + |    value:
     14 + |      apps.tanzu.vmware.com/pipeline: jenkins-pipeline
     15 + |  - name: testing_pipeline_params
     16 + |    value:
     17 + |      job-name: jenkins-job
     18 + |      job-params:
     19 + |      - name: param1
     20 + |        value: value1
     21 + |      secret-name: my-secret
     22 + |  source:
     23 + |    git:
     24 + |      ref:
```

```
25 + |        branch: my-branch
26 + |      url: https://my-source-code-repository
```

Building container images with Supply Chain Choreographer

This topic describes the methods you can use to build container images for Supply Chain Choreographer for Tanzu.

Methods for building container images

You can build a container image by using:

- A Maven artifact. See Building from source
- A Dockerfile based build. See Dockerfile-based builds
- Tanzu Build Service with buildpacks. See Tanzu Build Service Integration

Building from source with Supply Chain Choreographer

You can build from source by providing source code for the workload with any Supply Chain package.

You can provide source code for the workload from one of three places:

1. A Git repository.
2. A directory in your local computer's file system.
3. A Maven repository.

```
Supply Chain

-- fetch source                 * either from Git or local directory
  -- test
    -- build
      -- scan
        -- apply-conventions
          -- push config
```

This document provides details about each approach.

Note

To provide a prebuilt container image instead of building the application from the beginning by using the supply chain, see Using an existing image.

Git source

To provide source code from a Git repository to the supply chains, you must fill `workload.spec.source.git`. With the Tanzu CLI, you can do so by using the following flags:

- `--git-branch`: branch within the Git repository to checkout
- `--git-commit`: commit SHA within the Git repository to checkout
- `--git-repo`: Git URL to remote source code
- `--git-tag`: tag within the Git repository to checkout

For example, after installing `ootb-supply-chain-basic`, to create a `Workload` the source code for which comes from the `main` branch of the `github.com/vmware-tanzu/application-accelerator-samples` Git repository, and the subdirectory `tanzu-java-web-app` run:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --git-branch main
```

Expect to see the following output:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  source:
     12 + |    git:
     13 + |      ref:
     14 + |        branch: main
     15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     16 + |    subPath: tanzu-java-web-app
```

Important

The Git repository URL must include the scheme: `http://`, `https://`, or `ssh://`.

Private `GitRepository`

To fetch source code from a repository that requires credentials, you must provide those by using a Kubernetes secret object that the `GitRepository` object created for that workload references. See How It Works to learn more about detecting changes to the repository.

```
Workload/tanzu-java-web-app
└─GitRepository/tanzu-java-web-app
                  └────────> secretRef: {name: GIT-SECRET-NAME}
                                                  |
                                        either a default from TAP installation or
                                             gitops_ssh_secret Workload parameter
```

Platform operators who install the Out of the Box Supply Chain packages by using Tanzu Application Platform profiles can customize the default name of the secret (`git-ssh`) by editing the corresponding `ootb_supply_chain*` property in the `tap-values.yaml` file:

```
ootb_supply_chain_basic:
  gitops:
    ssh_secret: GIT-SECRET-NAME
```

For platform operators who install the `ootb-supply-chain-*` package individually by using `tanzu package install`, they can edit the `ootb-supply-chain-*-values.yml` as follows:

```
gitops:
  ssh_secret: GIT-SECRET-NAME
```

You can also override the default secret name directly in the workload by using the `gitops_ssh_secret` parameter, regardless of how Tanzu Application Platform is installed. You can use the `--param` flag in Tanzu CLI. For example:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --git-branch main \
  --param gitops_ssh_secret=SECRET-NAME
```

Expect to see the following output:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  params:
     12 + |  - name: gitops_ssh_secret  #! parameter that overrides the default
     13 + |    value: GIT-SECRET-NAME     #! secret name
     14 + |  source:
     15 + |    git:
     16 + |      ref:
     17 + |        branch: main
     18 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
     19 + |    subPath: tanzu-java-web-app
```

Note

A secret reference is only provided to `GitRepository` if `gitops_ssh_secret` is set to a non-empty string in some fashion, either by a package property or a workload parameter. To force a `GitRepository` to not reference a secret, set the value to an empty string (`""`).

After defining the name of the Kubernetes secret, you can define the secret.

HTTP(S) Basic-authentication and Token-based authentication

Despite both the package value and workload parameter being called `gitops.ssh_secret`, you can use HTTP(S) transports as well:

1. Ensure that the repository in the `Workload` specification uses `http://` or `https://` schemes in any URLs that relate to the repositories. For example, `https://github.com/my-org/my-repo` instead of `github.com/my-org/my-repo` or `ssh://github.com:my-org/my-repo`.
2. In the same namespace as the workload, create a Kubernetes secret object of type `kubernetes.io/basic-auth` with the name matching the one expected by the supply chain. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME
  annotations:
    tekton.dev/git-0: GIT-SERVER        # ! required
type: kubernetes.io/basic-auth
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD
```

3. With the secret created with the name matching the one configured for `gitops.ssh_secret`, attach it to the `ServiceAccount` used by the workload. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

For more information about the credentials and setting up the Kubernetes secret, see Git Authentication's HTTP section.

SSH authentication

Aside from using HTTP(S) as a transport, you can also use SSH:

1. Ensure that the repository URL in the workload specification uses `ssh://` as the scheme in the URL, for example, `ssh://git@github.com:my-org/my-repo.git`
2. Create a Kubernetes secret object of type `kubernetes.io/ssh-auth`:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME
  annotations:
    tekton.dev/git-0: GIT-SERVER
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY     # private key with push-permissions
  identity: SSH-PRIVATE-KEY           # private key with pull permissions
  identity.pub: SSH-PUBLIC-KEY        # public of the `identity` private key
  known_hosts: GIT-SERVER-PUBLIC-KEYS # git server public keys
```

3. With the secret created with the name matching the one configured for `gitops.ssh_secret`, attach it to the ServiceAccount used by the workload. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
```

```
  - name: registry-credentials
  - name: tap-registry
```

For information about how to generate the keys and set up SSH with the Git server, see Git Authentication's SSH section.

How it works

With the `workload.spec.source.git` filled, the supply chain takes care of managing a child `GitRepository` object that keeps track of commits made to the Git repository stated in `workload.spec.source.git`.

For each revision found, `gitrepository.status.artifact` gets updated providing information about an HTTP endpoint that the controller makes available for other components to fetch the source code from within the cluster.

The digest of the latest commit:

```
apiVersion: source.toolkit.fluxcd.io/v1beta1
kind: GitRepository
metadata:
  name: tanzu-java-web-app
spec:
  gitImplementation: go-git
  ignore: '!.git'
  interval: 1m0s
  ref: {branch: main}
  timeout: 20s
  url: https://github.com/vmware-tanzu/application-accelerator-samples
status:
  artifact:
    checksum: 375c2daee5fc8657c5c5b49711a8e94d400994d7
    lastUpdateTime: "2022-04-07T15:02:30Z"
    path: gitrepository/default/tanzu-java-web-app/d85df1fc.tar.gz
    revision: main/d85df1fc28c6b86ca54bd613f55991645d3b257c
    url: http://source-controller.flux-system.svc.cluster.local./gitrepository/defaul
t/tanzu-java-web-app/d85df1fc.tar.gz
  conditions:
  - lastTransitionTime: "2022-04-07T15:02:30Z"
    message: 'Fetched revision: main/d85df1fc28c6b86ca54bd613f55991645d3b257c'
    reason: GitOperationSucceed
    status: "True"
    type: Ready
  observedGeneration: 1
```

Cartographer passes the artifact URL and revision to further components in the supply chain. Those components must consume the source code from an internal URL where a tarball with the source code is fetched, without having to process any Git-specific details in multiple places.

Workload parameters

You can pass the following parameters by using the workload object's `workload.spec.params` field to override the default behavior of the `GitRepository` object created for keeping track of the changes to a repository:

- `gitImplementation`: name of the Git implementation (either `libgit2` or `go-git`) to fetch the source code.
- `gitops_ssh_secret`: name of the secret in the same namespace as the workload where credentials to fetch the repository are found.

You can also customize the following parameters with defaults for the whole cluster. Do this by using properties for either `tap-values.yaml` when installing supply chains by using Tanzu

Application Platform profiles, or `ootb-supply-chain-*-values.yml` when installing the OOTB packages individually):

- `git_implementation`: the same as `gitImplementation` workload parameter
- `gitops.ssh_secret`: the same as `gitops_ssh_secret` workload parameter

Local source

You can provide source code from a local directory such as, from a directory in the developer's file system. The Tanzu CLI provides two flags to specify the source code location in the file system and where the source code is pushed to as a container image:

- `--local-path`: path on the local file system to a directory of source code to build for the workload
- `--source-image`: destination image repository where source code is staged before being built

This way, whether the cluster the developer targets is local (a cluster in the developer's machine) or not, the source code is made available by using a container image registry.

For example, if a developer has source code under the current directory (`.`) and access to a repository in a container image registry, you can create a workload as follows:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --local-path . \
  --source-image $REGISTRY/test
```

```
Publish source in "." to "REGISTRY-SERVER/REGISTRY-REPOSITORY"?
It may be visible to others who can pull images from that repository

  Yes

Publishing source in "." to "REGISTRY-SERVER/REGISTRY-REPOSITORY"...
Published source

Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  source:
     12 + |    image: REGISTRY-SERVER/REGISTRY-REPOSITORY:latest@<digest>
```

Where:

- `REGISTRY-SERVER` is the container image registry.
- `REGISTRY-REPOSITORY` is the repository in the container image registry.

Authentication

Both the cluster and the developer's machine must be configured to properly provide credentials for accessing the container image registry where the local source code is published to.

Developer

The Tanzu CLI must push the source code to the container image registry indicated by `--source-image`. To do so, the CLI must find the credentials, so the developer must configure their machine accordingly.

To ensure credentials are available, use `docker` to make the necessary credentials available for the Tanzu CLI to perform the image push. Run:

```
docker login REGISTRY-SERVER -u REGISTRY-USERNAME -p REGISTRY-PASSWORD
```

Supply chain components

Aside from the developer's ability to push source code to the container image registry, the cluster must also have the proper credentials, so it can pull that container image, unpack it, run tests, and build the application.

To provide the cluster with the credentials, point the ServiceAccount used by the workload at the Kubernetes secret that contains the credentials.

If the registry that the developer targets is the same one for which credentials were provided while setting up the workload namespace, no further action is required. Otherwise, follow the same steps as recommended for the application image.

How it works

A workload specifies that source code must come from an image by setting `workload.spec.source.image` to point at the registry provided by using `--source-image`. Instead of having a GitRepository object created, an ImageRepository object is instantiated, with its specification filled in such a way to keep track of images pushed to the registry provided by the user.

Take the following workload as an example:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: app
  labels:
    app.kubernetes.io/part-of: app
    apps.tanzu.vmware.com/workload-type: web
spec:
  source:
    image: 10.188.0.3:5000/test:latest
```

Instead of a `GitRepository` object, an `ImageRepository` is created:

```
  Workload/app
  │
- ├─GitRepository/app
+ ├─ImageRepository/app
  │
  ├─Image/app
  │ ├─Build/app-build-1
  │ │ └─Pod/app-build-1-build-pod
  │ ├─PersistentVolumeClaim/app-cache
  │ └─SourceResolver/app-source
  │
  ├─PodIntent/app
  │
  ├─ConfigMap/app
```

```
  |
  └─Runnable/app-config-writer
    └─TaskRun/app-config-writer-2zj7w
      └─Pod/app-config-writer-2zj7w-pod
```

`ImageRepository` provides the same semantics as `GitRepository`, except that it looks for source code in container image registries rather than Git repositories.

Maven Artifact

This approach aids integration with existing CI systems, such as Jenkins, and can pull artifacts from existing Maven repositories, including Jfrog Artifactory.

There are no dedicated fields in the `Workload` resource for specifying the Maven artifact configuration. You must fill in the `name`/`value` pairs in the `params` structure.

For example:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: my-workload
  labels:
    apps.tanzu.vmware.com/workload-type: web
spec:
  params:
  - name: maven
    value:
      groupId: com.example
      artifactId: springboot-initial
      version: RELEASE      # latest 'RELEASE' or a specific version (e.g.: '1.2.2')
      type: jar             # optional (defaults to 'jar')
      classifier: sources   # optional
```

There are two ways to create a workload that defines a specific version of a Maven artifact as source in the Tanzu CLI.

The first way is to define the source through CLI flags. For example:

```
tanzu apps workload apply my-workload \
      --maven-artifact springboot-initial \
      --maven-version 2.6.0 \
      --maven-group com.example \
      --type web --app spring-boot-initial -y
```

Another flag that can be used alongside the others in this type of command is `--maven-type`, which refers to the Maven packaging type and defaults to `jar` if not specified.

The second one is through complex params (in JSON or YAML format). To specify the Maven info with this method, run:

```
tanzu apps workload apply my-workload \
      --param-yaml maven='{"artifactId": "springboot-initial", "version": "2.6.0", "gr
oupId": "com.example"}'\
      --type web --app spring-boot-initial -y
```

To create a workload that defines the `RELEASE` version of a maven artifact as source, run:

```
tanzu apps workload apply my-workload \
      --param-yaml maven='{"artifactId": "springboot-initial", "version": "RELEASE",
"groupId": "com.example"}'\
      --type web --app spring-boot-initial -y
```

The Maven repository URL and required credentials are defined in the supply chain, not the workload. For more information, see Installing OOTB Basic.

Maven Repository Secret

The MavenArtifact only supports authentication using basic authentication.

Additionally, MavenArtifact supports security using the TLS protocol. The Application Operator can configure the MavenArtifact to use a custom, or self-signed certificate authority (CA).

The MavenArtifact expects that all of the earlier credentials are provided in one secret, formatted as shown later:

```
---
apiVersion: v1
kind: Secret
metadata:
  name: maven-credentials
type: Opaque
data:
  username: <BASE64>  # basic auth user name
  password: <BASE64>  # basic auth password
  caFile: <BASE64>    # PEM Encoded certificate data for custom CA
```

You cannot use the Tanzu CLI to create secrets such as this, but you can use the kubectl CLI instead.

For example:

```
kubectl create secret generic maven-credentials \
  --from-literal=username=literal-username \
  --from-file=password=/path/to/file/with/password.txt \
  --from-file=caFile=/path/to/ca-certificate.pem
```

Use Dockerfile-based builds with Supply Chain Choreographer

This topic explains how you can use Dockerfile-based builds with Supply Chain Choreographer.

For any source-based supply chains, when you specify the new `dockerfile` parameter in a workload, the builds switch from using Kpack to using Kaniko. Source-based supply chains are supply chains that don't take a pre-built image. Kaniko is an open-source tool for building container images from a Dockerfile without running Docker inside a container.

Use Dockerfile-based builds with Supply Chain Choreographer

Parameter name	Description	Example
`dockerfile`	relative path to the Dockerfile file in the build context	`./Dockerfile`
`docker_build_context`	relative path to the directory where the build context is	`.`
`docker_build_extra_args`	list of flags to pass directly to Kaniko (such as providing arguments, and so on to a build)	`- --build-arg=MY_KEY=MY_VALUE`

To build a container image from the `github.com/my-foo/bar` repository where the Dockerfile resides in the root of that repository, you can switch from using Kpack to building from that Dockerfile by passing the `dockerfile` parameter:

```
$ tanzu apps workload create my-foo \
  --git-repo https://github.com/my-foo/bar \
  --git-branch dev \
  --param dockerfile=./Dockerfile \
  --type web

  Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |  name: my-foo
      8 + |  namespace: dev
      9 + |spec:
     10 + |  params:
     11 + |  - name: dockerfile
     12 + |    value: ./Dockerfile
     13 + |  source:
     14 + |    git:
     15 + |      ref:
     16 + |        branch: dev
     17 + |      url: https://github.com/my-foo/bar
```

Similarly, if the context to be used for the build must be set to a different directory within the repository, you can make use of the `docker_build_context` to change that:

```
$ tanzu apps workload create my-foo \
  --git-repo https://github.com/my-foo/bar \
  --git-branch dev \
  --param dockerfile=MyDockerfile \
  --param docker_build_context=./src
```

Important

This feature has no platform operator configurations to be passed through `tap-values.yaml`, but if `ootb-supply-chain-*.registry.ca_cert_data` or `shared.ca_cert_data` is configured in `tap-values`, the certificates are considered when pushing the container image.

OpenShift

Despite that Kaniko can perform container image builds without needing either a Docker daemon or privileged containers, it does require the use of:

- Capabilities usually dropped from the more restrictive SecurityContextConstraints (SCC) enabled by default in OpenShift.
- The root user.

To overcome such limitations imposed by the default unprivileged SecurityContextConstraints (SCC), Tanzu Application Platform installs:

- `SecurityContextConstraints/ootb-templates-kaniko-restricted-v2-with-anyuid` with enough extra privileges for Kaniko to operate.

- `ClusterRole/ootb-templates-kaniko-restricted-v2-with-anyuid` to permit the use of such SCC to any actor binding to that cluster role.

Each developer namespace needs a role binding that binds the role to an actor: `ServiceAccount`. For more information, see Set up developer namespaces to use your installed packages.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: workload-kaniko-scc
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ootb-templates-kaniko-restricted-v2-with-anyuid
subjects:
  - kind: ServiceAccount
    name: default
```

With the SCC created and the ServiceAccount bound to the role that permits the use of the SCC, OpenShift accepts the pods created to run Kaniko to build the container images.

Note

Such restrictions are due to well-known limitations in how Kaniko performs the image builds, and there is currently no solution. For more information, see kaniko#105.

Tanzu Build Service integration for Supply Chain Choreographer

This topic describes how you can configure and use the Tanzu Build Service integration for Supply Chain Choreographer.

By default, the Out of the Box supply chains (`ootb-supply-chain-*`) in Tanzu Application Platform make use of Tanzu Build Service for building container images out of source code.

You can configure a **platform operator** by using `tap-values.yaml`:

1. The default container image registry where application images must be pushed:

   ```
   ootb_supply_chain_basic:
     registry:
       server: <>
       repository: <>
   ```

2. The name of the Kpack `ClusterBuilder` used by default:

   ```
   ootb_supply_chain_basic:
     cluster_builder: my-custom-cluster-builder
   ```

You can configure an **application operator** by using `Workload`:

- `spec.build.env` are the environment variables used during the build:

  ```
  kind: Workload
  apiVersion: carto.run/v1alpha1
  metadata:
  name: tanzu-java-web-app
  spec:
  # ...
  ```

```
build:
  env:
    - name: PORT
      value: "8080"
    - name: CA_CERTIFICATE
      valueFrom:
        secretKeyRef:
          name: secret-in-the-same-namespace-as-workload
          key: crt.pem
```

- `spec.params.clusterBuilder` is the name of the ClusterBuilder to use for builds of that Workload:

```
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
name: tanzu-java-web-app
spec:
# ...
params:
  - name: clusterBuilder
    value: nodejs-cluster-builder
```

- `spec.params.buildServiceBindings` is the object carrying the definition of a list of service bindings to use at build time:

```
---
kind: Workload
apiVersion: carto.run/v1alpha1
metadata:
name: tanzu-java-web-app
spec:
# ...
params:
  - name: buildServiceBindings
    value:
      - name: settings-xml
        kind: Secret
        apiVersion: v1
---
apiVersion: v1
kind: Secret
metadata:
name: settings-xml
type: service.binding/maven
stringData:
type: maven
provider: sample
settings.xml: <settings>...</settings>
```

Note

See the Kpack ServiceBinding documentation in GitHub for more details about build-time service bindings.

these configuration only take effect when Kpack is used for building a container image. If you use Dockerfile-based builds by leveraging the `dockerfile` parameter, see dockerfile-based builds for more information.

Configure and deploy to multiple environments with custom parameters

This topic describes how to use carvel packages, Git repositories, and Flux CD to deploy workloads to multiple environments with Supply Chain Choreographer. By using a continuous delivery (CD) tool, you can apply Carvel packages to a runtime.

Flux CD is the VMware recommended CD tool. You can configure different parameters for each environment, such as replicas or host names. When you edit package parameters and commit them to a Git repository, Flux CD watches the Git repository and applies the package to your runtime environments.

Feature limits

To configure and deploy to multiple environments with custom parameters, ensure that your supply chains are compatible with the feature limits.

This feature is in alpha and has the following limits:

- Only the Out of the Box Basic Supply Chain package is supported.
- The Testing and Scanning supply chains are not supported.
- Only workloads of type server are supported.
- Innerloop development is not supported.
- Azure GitOps repositories are not supported.

Using Carvel packages

You can configure your supply chain to outputs Carvel packages and deliver configuration for each environment. For information about using Carvel, see Carvel Package Supply Chains (alpha).

Using GitOps delivery with Flux CD

You can deliver packages created by the Carvel package supply chain, and add them to clusters, by using a GitOps repository. For information about this delivery method, see Use Gitops Delivery with Flux CD (alpha).

Using GitOps delivery with Carvel App

Alternatively, you can deliver packages created by the Carvel package supply chain, and add them to clusters by using a GitOps repository. For information about this delivery method, see Use Gitops Delivery with Carvel App

Configuring blue-green deployment

You can use blue-green deployment to transfer user traffic from one version of an app to a later version while both are running. For information about setting up blue-green deployment, see Use blue-green deployment with Contour and PackageInstall (alpha).

Carvel Package Supply Chains (alpha)

This topic explains what Carvel Package Supply Chains do, how they work, how operators can enable them, and how to create a `Workload` that uses them. You can use the Carvel Package

Supply Chains with Supply Chain Choreographer to deliver applications to multiple production environments with configuration for each environment.

The Out of the Box Basic Supply Chain package introduces a variation of the OOTB Basic supply chains that outputs Carvel Packages.

Overview of the Carvel Package Supply Chains

The out of the box Basic Supply Chain outputs a `Deliverable` object. These `Deliverable`s are deployed to a cluster by the Out of the Box Delivery Supply Chain. The Carvel Package Supply Chains output a Carvel `Package` object to a GitOps repository. These `Packages` have configurable parameters such as `hostname` and `replicas` that are configured per environment. GitOps tools such as Flux CD and Argo CD can deploy the `Package`s onto multiple environments.

Note

The underlying Kubernetes resources created for your `server Workload` are the same for `source-to-url` or `basic-image-to-url`, with the addition of a `networking.k8s.io/v1 Ingress` resource. For example, `Deployment`. The Carvel `Package` wraps these resources.

What do the Carvel Package Supply Chains Do?

There are two Carvel Package Supply Chains, `source-to-url-package` and `basic-image-to-url-package` in the Out of the Box Supply Chain Package. They are identical to `source-to-url` and `basic-image-to-url`, except for three resources:

- A new `carvel-package` resource is added. The supply chain stamps out a Tekton Task that bundles all application Kubernetes resources into a Carvel `Package`.
- `config-writer` is modified to write the Carvel `Package` to a GitOps repository.
- `deliverable` resource is removed.

When a `Workload` is created and all Supply Chain resources are stamped out, a Carvel `Package` is written to the GitOps repository at the path `<package_name>/packages/<package_id>.yaml`. `<package_name>` defaults to `<workload_name>.<workload_namespace>.tap`, and is customized with the `name_suffix` parameter. `<package_id>` is a SemVer compatible version generated by the Bash command `$(date "+%Y%m%d%H%M%S.0.0")`.

For example:

```
app.default.tap/
  packages/
    20230321004057.0.0.yaml
```

Note

By default, the `<package_name>` directory is created in the root directory of the GitOps repository. You can optionally create this directory at a subpath by configuring the `gitops_subpath` parameter.

For example, the following Carvel `Package` definition is stored in `<package_id>.yaml`:

```
apiVersion: data.packaging.carvel.dev/v1alpha1
kind: Package
```

```
metadata:
  name: app.default.tap.20230321004057.0.0
spec:
  refName: app.default.tap
  version: 20230321004057.0.0
  releaseNotes: |
    Release v20230321004057.0.0 of package app.default.tap
  template:
    spec:
      fetch:
      - imgpkgBundle:
          image: # imgpkg bundle containing all Kubernetes configuration
      template:
      - ytt:
          paths:
          - .
      - kbld:
          paths:
          - .imgpkg/images.yml
          - '-'
      deploy:
      - kapp: {}
```

The Carvel `Package` generated by the Supply Chain has three configurable parameters:

- `replicas`: Number of pods that you want for the `apps/v1 Deployment`. Default is `1`.
- `hostname`: Host name for the `networking.k8s.io/v1 Ingress`. Default is `example.com`.
- `port`: Port for the `networking.k8s.io/v1 Ingress`. Default is `8080`.

When a new commit is pushed to the source code Git Repository, such as `source-to-url-package`, or a new pre-built image is created, like `basic-image-to-url-package`, the Supply Chain stamps out a new version of the Carvel `Package`. This definition is written to `<package_name>/packages/<package_id>.yaml` with a new `<package_id>`.

The Carvel `Package` stored in GitOps repositories are deployed to multiple run clusters using GitOps tools, such as Flux CD or Argo CD. See Deploy Carvel Packages using Flux CD Kustomization.

Installing the Carvel Package Supply Chains as an Operator

This section describes operator tasks for enabling and configuring the Carvel Package Supply Chains.

Prerequisites

The Carvel Package Supply Chains require access to a GitOps repository and credentials. See GitOps versus RegistryOps.

Installation

In `tap-values`, configure the Out of the Box Basic Supply Chain package with the following parameters:

1. (Required) Enable the Carvel Package workflow.

   ```
   ootb_supply_chain_basic:
     carvel_package:
       workflow_enabled: true
   ```

2. (Optional) Set a GitOps subpath. This verifies the path in your GitOps repository where Carvel Packages are written. Defaults to `""`. See Template reference.

```
ootb_supply_chain_basic:
  carvel_package:
    workflow_enabled: true
    gitops_subpath: path/to/my/dir
```

3. (Optional) Set a name suffix. Carvel Package names are chosen using the `<workload_name>.<workload_namespace>.<name_suffix>` template. Defaults to `tap`. See Template reference.

```
ootb_supply_chain_basic:
  carvel_package:
    workflow_enabled: true
    name_suffix: vmware.com
```

4. Configure the Out of the Box Basic Supply Chain package with your GitOps parameters. See GitOps versus RegistryOps.
5. Install the Out of the Box Basic Supply Chain package.

Verifying the Carvel Package Supply Chains are Installed

1. Run `kubectl get ClusterSupplyChains`.
2. Confirm you see both `source-to-url-package` and `basic-image-to-url-package` with status `Ready: True`.

Using the Carvel Package Supply Chains as a Developer

This section describes developer tasks for using the Carvel Package Supply Chains.

Prerequisites

Your operator must install the Carvel Package Supply Chains.

You must create your workload in a developer namespace. See Developer namespace.

Creating a Workload

To use the Carvel Package Supply Chains, you must add the label `apps.tanzu.vmware.com/carvel-package-workflow=true` to your workload.

1. Use the `--label apps.tanzu.vmware.com/carvel-package-workflow=true` Tanzu CLI flag.

   For example:

```
tanzu apps workload create tanzu-java-web-app \
--namespace DEVELOPER_NAMESPACE \
--app tanzu-java-web-app \
--type server \
--label apps.tanzu.vmware.com/carvel-package-workflow=true \
--image springcommunity/spring-framework-petclinic
```

Expect to see the following output:

```
Create workload:
    1 + |---
    2 + |apiVersion: carto.run/v1alpha1
    3 + |kind: Workload
    4 + |metadata:
    5 + |  labels:
    6 + |    app.kubernetes.io/part-of: tanzu-java-web-app
    7 + |    apps.tanzu.vmware.com/carvel-package-workflow: "true"
    8 + |    apps.tanzu.vmware.com/workload-type: server
```

```
 9 + |  name: tanzu-java-web-app
10 + |  namespace: DEVELOPER_NAMESPACE
11 + |spec:
12 + |  image: springcommunity/spring-framework-petclinic
```

1. (Optional) You can override parameters set by the operator. Set a GitOps subpath. This verifies the path in your GitOps repository to which Carvel Packages are written. Defaults to `""`. See Template referemce.

Set this parameter by modifying `workload.spec.params.carvel_package_gitops_subpath`. With the Tanzu CLI, you can do so by using the `--param carvel_package_gitops_subpath=path/to/my/dir` flag.

1. (Optional) Set a name suffix. Carvel Package names are chosen using the template `<workload_name>.<workload_namespace>.<name_suffix>`. Defaults to `tap`. See Template reference.

Set this parameter by modifying `workload.spec.params.carvel_package_name_suffix`. With the Tanzu CLI, you can do so by using the `--param carvel_package_name_suffix=vmware.com` flag.

1. (Optional) You can override GitOps parameters. See GitOps versus RegistryOps.

Verify the Carvel Package was Created

You now see a Carvel `Package` stored in your GitOps repository. For example, at the path `tanzu-java-web-app.default.tap/packages/20230321004057.0.0.yaml` you see a valid Carvel `Package` definition.

Next Steps

You can deploy the Carvel `Package` using tools such as Flux CD or Argo CD. See Deploy Carvel Packages using Flux CD Kustomization.

Use Gitops Delivery with a Carvel App (alpha)

This topic explains how you can deliver Carvel `Packages`, created by the Carvel Package Supply Chains, from a GitOps repository to one or more run clusters using Carvel App. You can use Carvel Package Supply Chains with Supply Chain Choreographer.

Prerequisites

To use GitOps Delivery with Carvel App, you must complete the following prerequisites:

- You must create a `Workload` that uses the Carvel Package supply chains. For information about Carvel Packages, see Carvel Package Supply Chains. You must have at least one Carvel `Package` generated by this `Workload` stored in your GitOps repository.
- You must have at least one Run cluster. Run clusters serve as your deployment environments. They can either be Tanzu Application Platform clusters, or Kubernetes clusters, but they must have kapp-controller and Contour installed. See the Carvel documentation and the Contour documentation.
- If you plan to use a build cluster to control the deployment on all of the run clusters, you must create a Build cluster that has network access to your run clusters. If you intend to deploy directly on the run cluster without using a build cluster, a build cluster is only necessary for building the package.

Set up Run cluster namespaces

Each Run cluster must have a namespace and `ServiceAccount` with the correct permissions to deploy the Carvel `Packages`.

To set up a developer namespace if your Run cluster is also a Tanzu Application Platform cluster, see Set up developer namespaces to use your installed packages.

If your Run cluster is not a Tanzu Application Platform cluster, create a namespace and `ServiceAccount` with the following permissions:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: RUN-CLUSTER-NS
  name: app-cr-role
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: [""]
  resources: ["configmaps", "services"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get", "list", "create", "update", "delete"]
```

Where `RUN-CLUSTER-NS` is the name of your run cluster you want to create a namespace with.

Create Carvel PackageInstalls and secrets

For each Carvel `Package` and each Run cluster, you must create a Carvel `PackageInstall` and a `Secret`. The Carvel `PackageInstall` and the `Secret` is stored in your GitOps repository and deployed to Run clusters by the Carvel `App`.

The following example shows GitOps repository structure after completing the procedures in this section:

```
app.default.tap/
  packages/
    20230321004057.0.0.yaml  # Package
  staging/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
  prod/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
```

1. For each Run cluster, create a `Secret` that has the values for each `Package` parameter. To see the configurable properties of the `Package`, inspect the `Package` CR's valuesSchema. See Carvel Package Supply Chains. Store the `Secret` in your GitOps repository at `PACKAGE-NAME/RUN-CLUSTER/params.yaml`.

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
    name: app-values
   stringData:
    values.yaml: |
      ---
      replicas: 2
      hostname: app.mycompany.com
   ```

Where:

- `PACKAGE-NAME` is the name of your Carvel package you want to use.
- `RUN-CLUSTER` is the name of the run cluster you want to use with the package.

Note

You can skip this step to use the default parameter values.

2. For each Run cluster, create a `PackageInstall`. Reference the `Secret` you created earlier. Store the `PackageInstall` in your GitOps repository at `PACKAGE-NAME/RUN-CLUSTER/packageinstall.yaml`.

```
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
 name: app
spec:
 serviceAccountName: RUN-CLUSTER-NS-SA # ServiceAccount on Run cluster with per
missions to deploy Package, see "Set up Run Cluster Namespaces"
 packageRef:
   refName: app.default.tap # name of the Package
   versionSelection:
     constraints: 20230321004057.0.0 # version of the Package
 values:
 - secretRef:
     name: app-values # Secret created in previous step
```

Where:

- `PACKAGE-NAME` is the name of your Carvel package you want to use.
- `RUN-CLUSTER` is the name of the run cluster you want to use with the package.
- `RUN-CLUSTER-NS-SA` is the ServiceAccount on your run cluster with permissions to deploy the package.

Note

To continuously deploy the latest version of your `Package`, set `versionSelection.constraints: >=0.0.0`.

Important If you skipped creation of the `Secret`, omit the `values` key.

3. Push the newly created `PackageInstalls` and `Secrets` to your GitOps repository.

Create an App

1. You must give the Build cluster access to the Run clusters. On the Build cluster, for each Run cluster, create a `Secret` containing the Run cluster's kubeconfig:

```
kubectl create secret generic RUN-CLUSTER-kubeconfig \
   -n BUILD-CLUSTER-NS \
   --from-file=value.yaml=PATH-TO-RUN-CLUSTER-KUBECONFIG
```

Where:

- `RUN-CLUSTER` is the name of the run cluster you want to use with your app.

- `BUILD-CLUSTER-NS` is the namespace of the build cluster you want to use.
- `PATH-TO-RUN-CLUSTER-KUBECONFIG` is the location of your run cluster kubeconfig.

2. Each Carvel `App` custom resource (CR) must specify either a service account, by using `spec.serviceAccountName`, in the same namespace where the App CR is located on the Build cluster. Or specify a `Secret` with kubeconfig contents for a target destination Run cluster, by using `spec.cluster.kubeconfigSecretRef.name`, to explicitly provide the needed privileges for managing app resources. The example in this section uses a target Run cluster.

3. The Carvel App custom resource represents a collection of Kubernetes resources that kapp-controller can fetch and deploy to a cluster. The `App` points at the Git repository branch where kapp-controller resources, such as `PackageRepository` and `Packages`, are defined. By default, an `App` custom resource syncs the cluster with its fetch source every 30 seconds to prevent the cluster state from drifting from its source of truth, which is a Git repository in this case. Create the following `App` on your Build cluster:

```
---
apiVersion: kappctrl.k14s.io/v1alpha1
kind: App
metadata:
 name: hello-app-app
 namespace: BUILD-CLUSTER-NS
spec:
 # specifies that app should be deployed to destination cluster;
 # by default, cluster is same as where this resource resides
 cluster:
   # specifies namespace in destination cluster
   namespace: ns2
   # specifies secret containing kubeconfig
   kubeconfigSecretRef:
     # specifies secret name within app's namespace
     name: cluster1
     # specifies key that contains kubeconfig
     key: value
 fetch:
 - git:
     url: # GitOps repo URL ex: https://github.com/mycompany/my-gitops
     ref: # GitOps repo branchex: origin/main
     subPath: PATH-FOR-PACKAGES # ex: hello-app.dev.tap/packages/
 - git:
     url: # GitOps repo URL ex: https://github.com/mycompany/my-gitops
     ref: # GitOps repo branch ex: origin/main
     subPath: PATH-FOR-PACKAGE-INSTALLS # ex: hello-app.dev.tap/runcluster1
 template:
 - ytt: {}

  deploy:
 - kapp:
     intoNs: DESIRED-NAMESPACE
     rawOptions: ["--dangerous-allow-empty-list-of-resources=true"]
```

Where:

- `DESIRED-NAMESPACE` is the namespace you want to use with your app.
- `PATH-FOR-PACKAGE-INSTALLS` is the package install path.
- `PATH-FOR-PACKAGES` is the package path.
- `BUILD-CLUSTER-NS` is the build cluster namespace.

Note

> The fetch section can includes entries for all the locations in the GitOps repository to deploy, and append with other run clusters if needed.

Verifying applications

To verify your installation:

1. Target a Run cluster. Confirm that all Packages from the GitOps repository are deployed:

   ```
   kubectl get packages -A
   ```

2. Target a Run cluster. Confirm that all PackageInstalls are reconciled:

   ```
   kubectl get packageinstalls -A
   ```

You can access your application on each Run cluster.

Use Gitops Delivery with Flux CD (alpha)

This topic explains how you can deliver Carvel `Packages`, created by the Carvel Package Supply Chains, from a GitOps repository to one or more run clusters using Flux CD and Supply Chain Choreographer.

Prerequisites

To use Gitops Delivery with Flux CD, you must complete the following prerequisites:

- You must create a `Workload` that uses either the `source-to-url-package` or `basic-image-to-url-package` Carvel Package Supply Chain. See the Carvel documentation. You must have at least one Carvel `Package` generated by this `Workload` stored in your GitOps repository.
- You must have at least one run cluster. run clusters serve as your deployment environments. They can either be Tanzu Application Platform clusters, or regular Kubernetes clusters, but they must have kapp-controller and Contour installed. See the Carvel documentation and the Contour documentation.
- If you want to use a build cluster to control the deployment on all the run clusters, you must create a build cluster that has network access to your run clusters. You must also ensure that you installed Flux CD Kustomize Controller. See the Flux documentation for installation instructions. If you intend to deploy directly on the run cluster without a build cluster, a build cluster is only necessary for building the package.

Set up run cluster namespaces

Each run cluster must have a namespace and `ServiceAccount` with the correct permissions to deploy the Carvel `Packages`.

If your run cluster is a Tanzu Application Platform cluster, see Set up developer namespaces to use your installed packages.

If your run cluster is not a Tanzu Application Platform cluster, create a namespace and `ServiceAccount` with the following permissions:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
```

```
metadata:
  namespace: <run-cluster-ns>
  name: app-cr-role
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: [""]
  resources: ["configmaps", "services"]
  verbs: ["get", "list", "create", "update", "delete"]
- apiGroups: ["networking.k8s.io"]
  resources: ["ingresses"]
  verbs: ["get", "list", "create", "update", "delete"]
```

Create Carvel PackageInstalls and secrets

For each Carvel `Package` and for each run cluster, you must create a Carvel `PackageInstall` and a `Secret`. The Carvel `PackageInstall` and the `Secret` is stored in your GitOps repository and deployed to run clusters by Flux CD.

The following example shows GitOps repository structure after completing this section:

```
app.default.tap/
  packages/
    20230321004057.0.0.yaml  # Package
  staging/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
  prod/
    packageinstall.yaml      # PackageInstall
    params.yaml              # Secret
```

1. For each run cluster, create a `Secret` that has the values for each `Package` parameter. You can see the configurable properties of the `Package` by inspecting the `Package` CR's valuesSchema, or in the Carvel Package Supply Chains documentation. Store the `Secret` in your GitOps repository at `<package_name>/<run_cluster>/params.yaml`.

   ```
   ---
   apiVersion: v1
   kind: Secret
   metadata:
    name: app-values
   stringData:
    values.yaml: |
      ---
      replicas: 2
      hostname: app.mycompany.com
   ```

   Note

   You can skip this step to use the default parameter values.

2. For each run cluster, create a `PackageInstall`. Reference the `Secret` you created earlier. Store the `PackageInstall` in your GitOps repository at `<package_name>/<run_cluster>/packageinstall.yaml`.

   ```
   ---
   apiVersion: packaging.carvel.dev/v1alpha1
   kind: PackageInstall
   ```

```
metadata:
 name: app
spec:
 serviceAccountName: <run-cluster-ns-sa> # ServiceAccount on run cluster with p
ermissions to deploy Package, see "Set up run Cluster Namespaces"
 packageRef:
   refName: app.default.tap # name of the Package
   versionSelection:
     constraints: 20230321004057.0.0 # version of the Package
 values:
 - secretRef:
     name: app-values # Secret created in previous step
```

Note

To continuously deploy the latest version of your `Package`, you can set `versionSelection.constraints: >=0.0.0` If you skipped creation of the `Secret`, omit the `values` key.

3. Push the newly created `PackageInstalls` and `Secrets` to your GitOps repository.

Create Flux CD GitRepository and Flux CD Kustomizations on the Build Cluster

Configure Flux CD on the Build cluster to deploy your `Packages`, `PackageInstalls`, and `Secrets` to each of your run clusters.

1. Give your Build cluster access to your run clusters. On the Build cluster, for each run cluster, create a `Secret` containing the run cluster's kubeconfig:

```
kubectl create secret generic <run-cluster>-kubeconfig \
   -n <build-cluster-ns> \
   --from-file=value.yaml=<path-to-run-cluster-kubeconfig>
```

2. Configure your Build cluster to clone the GitOps repository. On the Build cluster, create the following Flux CD `GitRepository`:

```
---
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: GitRepository
metadata:
 name: <package-name>-gitops-repo
 namespace: <build-cluster-ns>
spec:
 url: # GitOps repo URL
 gitImplementation: go-git
 ignore: |
   !.git
 interval: 30s
 ref:
   branch: # GitOps repo branch
 timeout: 60s

 # only required if GitOps repo is private (recommended)
 secretRef:
   name: <package-name>-gitops-auth
   namespace: <build-cluster-ns>

# only required if GitOps repo is private (recommended)
---
```

```
apiVersion: v1
kind: Secret
metadata:
 name: <package-name>-gitops-auth
 namespace: <build-cluster-ns>
type: Opaque
data:
 username: # base64 encoded GitHub (or other git remote) username
 password: # base64 encoded GitHub (or other git remote) personal access token
```

3. Configure your Build cluster to deploy your `Package` to the run clusters. For each run cluster, on the Build cluster, create the following Flux CD `Kustomization`:

```
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
 name: <package-name>-<run-cluster>-packages
 namespace: <build-cluster-ns>
spec:
 sourceRef:
   kind: GitRepository
   name: <package-name>-gitops-repo
   namespace: <build-cluster-ns>
 path: "./<package-name>/packages"
 interval: 5m
 timeout: 5m
 prune: true
 wait: true

 # where to deploy
 kubeConfig:
   secretRef:
     name: <run-cluster>-kubeconfig
     namespace: <build-cluster-ns>
 targetNamespace: <run-cluster-ns>
 serviceAccountName: <run-cluster-ns-sa>
```

4. Configure your Build cluster to deploy your `PackageInstalls` and `Secrets` to the run clusters. For each run cluster, on the Build cluster, create the following Flux CD `Kustomization`:

```
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
 # for the second run cluster, for example hello-app-prod2-packages
 name: <package-name>-<run-cluster>-packageinstalls
 namespace: <build-cluster-ns>
spec:
 sourceRef:
   kind: GitRepository
   name: <package-name>-gitops-repo
   namespace: <build-cluster-ns>
 path: "./<package-name>/<run-cluster>"
 interval: 5m
 timeout: 5m
 prune: true
 wait: true

 # where to deploy
 kubeConfig:
   secretRef:
     name: <run-cluster>-kubeconfig
```

```
    namespace: <build-cluster-ns>
targetNamespace: <run-cluster-ns>
serviceAccountName: <run-cluster-ns-sa>
```

Verifying Installation

To verify your installation:

1. On your Build cluster, confirm that your Flux CD GitRepository and Kustomizations are reconciling:

   ```
   kubectl get gitrepositories,kustomizations -A
   ```

2. Target a run cluster. Confirm that all Packages from the GitOps repository are deployed:

   ```
   kubectl get packages -A
   ```

3. Target a run cluster. Confirm that all PackageInstalls are reconciled:

   ```
   kubectl get packageinstalls -A
   ```

Now you can access your application on each run cluster.

Use blue-green deployment with Contour and PackageInstall for Supply Chain Choreographer (alpha)

Blue-green deployment is an application delivery model that lets you gradually transfer user traffic from one version of your app to a later version while both are running in production. This topic outlines how to use blue-green deployment with Packages and PackageInstalls.

Prerequisites

To use blue-green deployment, you must complete the following prerequisites:

- Complete the prerequisites in Configure and deploy to multiple environments with custom parameters.
- Configure Carvel for your supply chain. See Carvel Package Supply Chains (alpha).
- Configure Flux CD for your supply chain. See Deploy Package and PackageInstall using Flux CD Kustomization.

Add HTTPProxy to the blue deployment

The following example deploys a sample application, `hello-app`, to production using a Carvel Package and PackageInstall.

1. Create a Contour HTTPProxy resource to route traffic to the `hello-app` service from the URL `www.hello-app.mycompany.com`.

   ```
   apiVersion: projectcontour.io/v1
   kind: HTTPProxy
   metadata:
   name: www
   namespace: prod
   spec:
   virtualhost:
     fqdn: www.hello-app.mycompany.com
   routes:
   ```

```
- conditions:
  - prefix: /
  services:
  - name: hello-app
    port: 8080
```

Note

The services names used in HTTPProxy has to match the names of existing services. In this case, the name `hello-app` matches the service installed by the PackageInstall.

2. Apply the HTTPProxy to your cluster:

```
kubectl apply -f httpproxy.yaml
```

3. Verify that the HTTPProxy is present and the route serves traffic to your app.

```
kubectl get HTTPProxy --namespace=prod
```

This displays a list of all the HTTPproxies in the current namespace with their current names.

```
kubectl get HTTPProxy --namespace=prod
NAMESPACE         NAME                      FQDN
TLS SECRET              STATUS    STATUS DESCRIPTION
prod              www                       www.hello-app.mycompany.com
hello-app-cert          valid     Valid HTTPProxy
```

Create the green deployment

After a new version of the package is added to the GitOps repository, create a new PackageInstall for v1.0.1 to create the green deployment.

1. Create a `green-secret.yaml` file with a secret that contains the following ytt overlay.

```
---
apiVersion: v1
kind: Secret
metadata:
name: green-overlay-secret
namespace: prod
stringData:
custom-package-overlay.yaml: |
  #@ load("@ytt:overlay", "overlay")

  #@ kd = overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"})
  #@ ks = overlay.subset({"apiVersion":"v1", "kind": "Service"})
  #@ ki = overlay.subset({"apiVersion":"networking.k8s.io/v1", "kind": "Ingres
s"})
  #@ na = overlay.subset({"metadata":{"name":"hello-app"}})

  #@overlay/match by=overlay.and_op(kd, na)
  ---
  metadata:
    #@overlay/replace
    name: hello-app-green

  #@overlay/match by=overlay.and_op(ks, na)
  ---
```

```
  metadata:
    #@overlay/replace
    name: hello-app-green

  #@overlay/match by=overlay.and_op(ki, na)
  ---
  metadata:
    #@overlay/replace
    name: hello-app-green
```

This secret changes the names of the service and deployment in the Carvel Package to allow you to install another version of the app in the same namespace.

2. Apply the secret to your cluster by running:

```
kubectl apply -f green-secret.yaml
```

3. Create a parameter secret for the new PackageInstall:

```
---
apiVersion: v1
kind: Secret
metadata:
name: green-dev-values
namespace: prod
stringData:
values.yaml: |
  ---
  replicas: 2
  hostname: hello-app-green.mycompany.com
```

4. Apply the parameter secret to your cluster by running:

```
kubectl apply -f green-dev-values.yaml
```

5. Create a PackageInstall to include the `ext.packaging.carvel.dev/ytt-paths-from-secret-name.x` annotation to reference your new overlay secret.

```
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
name: green.hello-app.dev.tap
namespace: prod
annotations:
  ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: green-overlay-secret
spec:
serviceAccountName: default
packageRef:
  refName: hello-app.dev.tap
  versionSelection:
    constraints: "1.0.1"
values:
- secretRef:
    name: green-dev-values
```

Divide traffic between the blue and green deployments

Use the following procedure to divide traffic between your blue and green deployments.

1. Update the HTTPproxy created with the blue deployment to route traffic to both the blue and green deployments. The names of the services must match the names of the already

created services.

```
---
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
name: www
namespace: prod
spec:
virtualhost:
  fqdn: www.hello-app.mycompany.com
routes:
- conditions:
  - prefix: /
  services:
  - name: hello-app-green
    port: 8080
    weight: 20
  - name: hello-app
    port: 8080
    weight: 80
```

2. Update the weights of traffic for each service by editing the HTTPProxy.
3. Access the service several times and confirm both versions are serving traffic in the same percentage.

```
curl -k https://www.hello-app.mycompany.com
```

After the new green app is ready to handle the complete load and the `-green` version is not required, use the following steps to remove the old version and rename the new version:

1. Ensure that all the traffic is using the correct version of the app. For example:

```
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: www
  namespace: prod
spec:
  virtualhost:
    fqdn: www.hello-app.mycompany.com
  routes:
    - conditions:
      - prefix: /
      services:
        - name: hello-app-green
          port: 8080
          weight: 100 # all traffic routed to the green app
```

2. Identify the name of the deployment and service that are part of the PackageInstall you no longer need:

```
kubectl get PackageInstall --namespace=prod
```

This displays a list of all the deployments and services in the current Kubernetes namespace, with their current names. For example:

```
NAME                      PACKAGE NAME       PACKAGE VERSION      DESCRIPTION
green.hello-app.dev.tap   hello-app.dev.tap   1.0.1            Reconcile succee
ded
hello-app.dev.tap         hello-app.dev.tap   1.0.0            Reconcile succee
```

```
ded
```

3. Delete the PackageInstall:

```
kubectl delete PackageInstall hello-app.dev.tap --namespace=prod
```

4. Rename the service and deployments without the green prefix. For example, update the overlay secret:

```
---
apiVersion: v1
kind: Secret
metadata:
name: overlay-secret
namespace: prod
stringData:
custom-package-overlay.yaml: |
  #@ load("@ytt:overlay", "overlay")
  #@ load("@ytt:data", "data")

  #@ kd = overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"})
  #@ ks = overlay.subset({"apiVersion":"v1", "kind": "Service"})
  #@ ki = overlay.subset({"apiVersion":"networking.k8s.io/v1", "kind": "Ingres
s"})
  #@ na = overlay.subset({"metadata":{"name":"hello-app-green"}})

  #@overlay/match by=overlay.and_op(kd, na)
  ---
  metadata:
    #@overlay/replace
    name: hello-app

  #@overlay/match by=overlay.and_op(ks, na)
  ---
  metadata:
    #@overlay/replace
    name: hello-app

  #@overlay/match by=overlay.and_op(ki, na)
  ---
  metadata:
    #@overlay/replace
    name: hello-app

  ---
  apiVersion: projectcontour.io/v1
  kind: HTTPProxy
  metadata:
    name: www
    namespace: prod
  spec:
    virtualhost:
      fqdn: www.hello-app.mycompany.com
    routes:
      - conditions:
        - prefix: /
        services:
          - name: hello-app # note the name is changed back
            port: 8080
            weight: 100
```

5. Update your PackageInstall to include the `ext.packaging.carvel.dev/ytt-paths-from-secret-name.x` annotation to reference your new overlay secret. For example:

```
---
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
name: green.hello-app.dev.tap
namespace: prod
annotations:
  ext.packaging.carvel.dev/ytt-paths-from-secret-name.0: overlay-secret
spec:
serviceAccountName: default
packageRef:
  refName: hello-app.dev.tap
  versionSelection:
    constraints: "1.0.1"
values:
- secretRef:
    name: hello-app-values
```

6. After the deployment is complete, you can delete the secrets with the overlays.

Verify application

To verify the name of the deployment and service that are part of the PackageInstall:

1. Verify your application by running:

   ```
   kubectl get PackageInstall --namespace=prod
   ```

   This displays a list of all the deployments and services in the current Kubernetes namespace with their current names. For example:

   ```
   NAME                     PACKAGE NAME        PACKAGE VERSION      DESCRIPTION
   hello-app.dev.tap        hello-app.dev.tap     1.0.1            Reconcile succee
   ded
   ```

   The name is back to the original name and the version is `1.0.1`.

Use an existing image with Supply Chain Choreographer

This topic describes how you can use an existing image with Supply Chain Choreographer.

For apps that build container images in a predefined way, the supply chains in the Out of the Box packages enable you to specify a prebuilt image. This uses the same stages as any other workload.

Requirements for prebuilt images

Supply chains aim at Knative as the runtime for the container image you provide. Your app must adhere to the following Knative standards:

- **Container port listens on port 8080**

  The Knative service is created with the container port set to `8080` in the pod template spec Therefore, your container image must have a socket listening on `8080`.

  ```
  ports:
    - containerPort: 8080
      name: user-port
      protocol: TCP
  ```

- **Non-privileged user ID**

By default, the container initiated as part of the pod is run as user 1000.

```
securityContext:
  runAsUser: 1000
```

- **Arguments other than the image's default ENTRYPOINT**

  In most cases the container image runs using the `ENTRYPOINT` it was configured with. In the case of Dockerfiles, the combination of `ENTRYPOINT` and `CMD`.

  If you need extra configuration for your image, use `--env` flags with the `tanzu apps workload create` command or modify `spec.env` in your `workload.yaml` file.

- **Credentials for pulling the container image at runtime**

  The image you provide is not relocated to an internal container image registry. Any components associated with the image must have the necessary credentials to pull it. For the service accounts used for the workload and deliverable, you must attach a secret that contains the credentials to pull the container image.

  If the image is hosted in a registry that has certificates signed by a private certificate authority, the components of the supply chains, delivery, and the Kubernetes nodes in the run cluster must trust the certificate.

Configure your workload to use a prebuilt image

To select a prebuilt image, set the `spec.image` field in your `workload.yaml` file with the name of the container image that contains the app to deploy by running:

```
tanzu apps workload create WORKLOAD-NAME \
  --app APP-NAME \
  --type TYPE \
  --image IMAGE
```

Where:

- `WORKLOAD-NAME` is the name you choose for your workload.
- `APP-NAME` is the name of your app.
- `TYPE` is the type of your app.
- `IMAGE` is the container image that contains the app you want to deploy.

For example, if you have an image named `IMAGE`, you can create a workload with the flag mentioned earlier:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --image IMAGE
```

Expected output:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: hello-world
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
```

```
 9 + |  namespace: default
10 + |spec:
11 + |  image: IMAGE
```

When you run `tanzu apps workload create` command with the `--image` field, the source resolution and build phases of the supply chain are skipped.

Examples

The following examples show ways that you can build container images for a Java-based app and complete the supply chains to a running service.

Using a Dockerfile

Using a Dockerfile is the most common way of building container images. You can select a base image, on top of which certain operations must occur, such as compiling code, and mutate the contents of the file system to a final container image that has a build of your app and any required runtime dependencies.

Here you use the `maven` base image for compiling your app code, and then the minimal distroless `java17-debian11` image for providing a JRE that can run your app when it is built.

After building the image, you push it to a container image registry, and then reference it in the workload.

1. Create a Dockerfile that describes how to build your app and make it available as a container image:

   ```
   ARG BUILDER_IMAGE=maven
   ARG RUNTIME_IMAGE=gcr.io/distroless/java17-debian11


   FROM $BUILDER_IMAGE AS build

           ADD . .
           RUN unset MAVEN_CONFIG && ./mvnw clean package -B -DskipTests


   FROM $RUNTIME_IMAGE AS runtime

           COPY --from=build /target/demo-0.0.1-SNAPSHOT.jar /demo.jar
           CMD [ "/demo.jar" ]
   ```

2. Push the container image to a container image registry by running:

   ```
   docker build -t IMAGE .
   docker push IMAGE
   ```

3. Create a workload by running:

   ```
   tanzu apps workload create tanzu-java-web-app \
     --type web \
     --app tanzu-java-web-app \
     --image IMAGE
   ```

   Expected output:

   ```
   Create workload:
         1 + |---
         2 + |apiVersion: carto.run/v1alpha1
         3 + |kind: Workload
   ```

```
      4 + |metadata:
      5 + |  labels:
      6 + |    app.kubernetes.io/part-of: hello-world
      7 + |    apps.tanzu.vmware.com/workload-type: web
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
     10 + |spec:
     11 + |  image: IMAGE
```

4. Run the following workload:

```
tanzu apps workload get tanzu-java-web-app
```

Expected output:

```
# tanzu-java-web-app: Ready
---
lastTransitionTime: "2022-04-06T19:32:46Z"
message: ""
reason: Ready
status: "True"
type: Ready

Workload pods
NAME                                                  STATUS      RESTARTS   A
GE
tanzu-java-web-app-00001-deployment-7d7df5ccf5-k58rt   Running     0          3
2s
tanzu-java-web-app-config-writer-xjmvw-pod            Succeeded   0          8
9s

Workload Knative Services
NAME                READY   URL
tanzu-java-web-app   Ready   http://tanzu-java-web-app.default.example.com
```

Using Spring Boot's `build-image` Maven target

You can use Spring Boot's `build-image` target to build a container image that runs your app. The `build-image` target must use a Dockerfile.

For example, using the same sample repository as mentioned before (https://github.com/vmware-tanzu/application-accelerator-samples/tree/main/tanzu-java-web-app):

1. Build the image by running the following command from the root of the repository:

```
IMAGE=ghcr.io/kontinue/hello-world:tanzu-java-web-app
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=$IMAGE
```

Expected output:

```
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------------< com.example:demo >--------------------------
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
...
[INFO]
[INFO] Successfully built image 'ghcr.io/kontinue/hello-world:tanzu-java-web-ap
p'
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

```
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  39.257 s
[INFO] Finished at: 2022-04-06T19:40:16Z
[INFO] ------------------------------------------------------------------------
```

2. Push the image you built to the container image registry by running:

   ```
   IMAGE=ghcr.io/kontinue/hello-world:tanzu-java-web-app
   docker push $IMAGE
   ```

   Expected output:

   ```
   The push refers to repository [ghcr.io/kontinue/hello-world]
   1dc94a70dbaa: Preparing
   ...
   9d6787a516e7: Pushed
   tanzu-java-web-app: digest: sha256:7140722ea396af69fb3d0ad12e9b4419bc3e67d9c5d8
   a2f6a1421decc4828ace size: 4497
   ```

After you push the container image, you see the same results as building the image using a Dockerfile.

For more information about building container images for a Spring Boot app, see Spring Boot with Docker

About Out of the Box Supply Chains

In Tanzu Application Platform, the `ootb-supply-chain-basic`, `ootb-supply-chain-testing`, and `ootb-supply-chain-testing-scanning` packages each receive a new supply chain that provides a prebuilt container image for your app.

```
ootb-supply-chain-basic

    (cluster)  basic-image-to-url   ClusterSupplyChain            (!) new
    ^          source-to-url        ClusterSupplyChain


ootb-supply-chain-testing

    (cluster)  testing-image-to-url  ClusterSupplyChain           (!) new
    ^          source-test-to-url    ClusterSupplyChain


ootb-supply-chain-testing-scanning

    (cluster)  scanning-image-scan-to-url    ClusterSupplyChain   (!) new
    ^          source-test-scan-to-url       ClusterSupplyChain
```

To leverage the supply chains that expect a prebuilt image, you must set the `spec.image` field in the workload to the name of the container image that contains the app to deploy.

The new supply chains use a Cartographer feature that lets VMware increase the specificity of supply chain selection by using the `matchFields` selector rule.

The selection takes place as follows:

- *ootb-supply-chain-basic*
  - From source: label `apps.tanzu.vmware.com/workload-type: web`
  - Prebuilt image: label `apps.tanzu.vmware.com/workload-type: web` **and** set `spec.image` in the `workload.yaml`

- *ootb-supply-chain-testing*
    - From source: labels `apps.tanzu.vmware.com/workload-type: web` and `apps.tanzu.vmware.com/has-tests: true`
    - Prebuilt image: label `apps.tanzu.vmware.com/workload-type: web` **and** set `spec.image` in the `workload.yaml`
- *ootb-supply-chain-testing-scanning*
    - From source: labels `apps.tanzu.vmware.com/workload-type: web` and `apps.tanzu.vmware.com/has-tests: true`
    - Prebuilt image: label `apps.tanzu.vmware.com/workload-type: web` **and** set `spec.image` in the `workload.yaml`

Workloads that already work with the supply chains before Tanzu Application Platform v1.1 continue to work with the same supply chain. Workloads that bring a prebuilt container image must set `spec.image` in the `workload.yaml`.

Understanding the supply chain for a prebuilt image

An `ImageRepository` object is created to keep track of new images pushed under that name. `ImageRepository` makes the image available to further resources in the supply chain, providing the final digest of the latest image.

Whenever a new image is pushed to the workload's image location, the `ImageRepository` detects the change. The image is then available to further resources by updating its `imagerepository.status.artifact.revision` with an absolute reference to that image.

For example, if you create a workload using an image named `hello-world`, tagged `tanzu-java-web-app` hosted under `ghcr.io` in the `kontinue` repository:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --image ghcr.io/kontinue/hello-world:tanzu-java-web-app
```

After a couple seconds, you see the `ImageRepository` object created to keep track of images named `ghcr.io/kontinue/hello-world:tanzu-java-web-app`:

```
Workload/tanzu-java-web-app
├─ImageRepository/tanzu-java-web-app
├─PodIntent/tanzu-java-web-app
├─ConfigMap/tanzu-java-web-app
└─Runnable/tanzu-java-web-app-config-writer
  └─TaskRun/tanzu-java-web-app-config-writer-p2lzv
    └─Pod/tanzu-java-web-app-config-writer-p2lzv-pod
```

If you inspect the status in `status.resources` in the `workload.yaml`, you see the `image-provider` resource promoting the image it found to further resources:

```
apiVersion: carto.run/v1alpha1
kind: Workload
spec:
  image: ghcr.io/kontinue/hello-world:tanzu-java-web-app
status:
  resources:
    - name: image-provider
      outputs:
        # output being made available to further resources in the supply chain
        # (in this case, the latest image it found under that name).
```

```
        #
        - name: image
          lastTransitionTime: "2022-04-01T15:05:01Z"
          preview: ghcr.io/kontinue/hello-world:tanzu-java-web-app@sha256:9fb930a...

      # reference to the object managed by the supply chain for this
      # resource
      #
      stampedRef:
        apiVersion: source.apps.tanzu.vmware.com/v1alpha1
        kind: ImageRepository
        name: tanzu-java-web-app
        namespace: workload

      # reference to the template that defined how this object should look
      # like
      #
      templateRef:
        apiVersion: carto.run/v1alpha1
        kind: ClusterImageTemplate
        name: image-provider-template
```

The image found by the `ImageRepository` object is carried through the supply chain to the final configuration. This is pushed to either a Git repository or image registry so that it is deployed in a run cluster.

Note

The image name matches the image name supplied in the `spec.image` field in the `workload.yaml`, but also includes the digest of the latest image found under the tag. If a new image is pushed to the same tag, you see the `ImageRepository` resolving the name to a different digest corresponding to the new image pushed.

Use Git authentication with Supply Chain Choreographer

This topic describes how you can use Git authentication with Supply Chain Choreographer.

You can either fetch or push source code from or to a repository that requires credentials. You must provide credentials through a Kubernetes secret object referenced by the intended Kubernetes object created for performing the action.

The following sections provide details about how to appropriately set up Kubernetes secrets for carrying those credentials forward to the proper resources.

Important

For both HTTP(s) and SSH, do not use the same server for multiple secrets to avoid a Tekton error.

HTTP

For any action upon an HTTP(s)-based repository, create a Kubernetes secret object of type `kubernetes.io/basic-auth` as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: SECRET-NAME
```

```
  annotations:
    tekton.dev/git-0: GIT-SERVER        # ! required
type: kubernetes.io/basic-auth          # ! required
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD
```

For example, assuming you have a repository called `kontinue/hello-world` on GitHub that requires authentication, and you have an access token with the privileges of reading the contents of the repository, you can create the secret as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: git-secret
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/basic-auth
stringData:
  username: GITHUB-USERNAME
  password: GITHUB-ACCESS-TOKEN
```

Note

In this example, you use an access token because GitHub deprecates basic authentication with plain user name and password. For more information, see Creating a personal access token on GitHub.

After you create the secret, attach it to the `ServiceAccount` configured for the workload by including it in its set of secrets. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

SSH

Aside from using HTTP(S) as a transport, the supply chains also allow you to use SSH.

Important

To use the pull request feature, you must use HTTP(S) authentication with an access token.

1. To provide the credentials for any Git operations with SSH, create the Kubernetes secret as follows:

   ```
   apiVersion: v1
   kind: Secret
   ```

```
metadata:
  name: GIT-SECRET-NAME
  annotations:
    tekton.dev/git-0: GIT-SERVER
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY     # private key with push-permissions
  identity: SSH-PRIVATE-KEY           # private key with pull permissions
  identity.pub: SSH-PUBLIC-KEY        # public of the `identity` private key
  known_hosts: GIT-SERVER-PUBLIC-KEYS # Git server public keys
```

2. Generate a new SSH keypair: `identity` and `identity.pub`.

```
ssh-keygen -t ecdsa -b 521 -C "" -f "identity" -N ""
```

3. Go to your Git provider and add the `identity.pub` as a deployment key for the repository of interest or add to an account that has access to it. For example, for GitHub, visit `https://github.com/<repository>/settings/keys/new`.

Note

Keys of type SHA-1/RSA are recently deprecated by GitHub.

4. Gather public keys from the provider, for example, GitHub:

```
ssh-keyscan github.com > ./known_hosts
```

5. Create the Kubernetes secret by using the contents of the files in the first step:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME
  annotations: {tekton.dev/git-0: GIT-SERVER}
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY
  identity: SSH-PRIVATE-KEY
  identity.pub: SSH-PUBLIC-KEY
  known_hosts: GIT-SERVER-PUBLIC-KEYS
```

For example, edit the credentials:

```
apiVersion: v1
kind: Secret
metadata:
  name: git-ssh
  annotations: {tekton.dev/git-0: github.com}
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    AAAA
    ....
    ....
    -----END OPENSSH PRIVATE KEY-----
  known_hosts: |
    <known hosts entrys for git provider>
  identity: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    AAAA
```

```
    ....
    ....
    -----END OPENSSH PRIVATE KEY-----
  identity.pub: ssh-ed25519 AAAABBBCCCCDDDDeeeeFFFF user@example.com
```

6. After you create the secret, attach it to the ServiceAccount configured for the workload by including it in its set of secrets. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

Read more on Git

For information about Git, see Git Reference.

Using Azure DevOps as a Git provider with your supply chains

This topic describes how you can Azure DevOps as a Git provider with your Supply Chain Choreographer supply chains.

Overview

There are two uses for Git in a supply chain:

- As a source of code to build and deploy applications
- As a repository of configuration created by the build cluster which is deployed on a run or production cluster

Azure DevOps differs from other Git providers in the following ways:

- Azure DevOps requires Git clients to support multi-ack.
- Azure DevOps repository paths differ from other Git providers.

For information about how Azure DevOps is different from other Git providers, see Gitops write path templates.

The operator requires special configuration to integrate Azure DevOps repositories into a supply chain.

Azure authentication

You can use Azure authentication with Supply Chain Choreographer.

For information about configuring secrets to authenticate with your Azure DevOps Git repository, see Use Git authentication with Supply Chain Choreographer.

Azure http and https authentication requires:

```
username: "_token"
password: AZURE-USER-TOKEN
```

Where `AZURE-USER-TOKEN` is your Azure personal access token. See Azure Devops Personal Access Tokens in the Microsoft documentation.

Using Azure DevOps as a repository for committed code

Developers can use Azure DevOps to commit source code to a repository that the supply chain pulls.

Azure DevOps example

The following example uses the Azure DevOps source repository:

`https://dev.azure.com/my-company/app/_git/app`

You can configure the supply chain by using `tap-values`:

```
ootb_supply_chain_testing_scanning:
  git_implementation: libgit2
```

or by using workload parameter:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  ...
spec:
  params:
    - name: gitImplementation
      value: libgit2
```

Configuring your Git implementation for Azure DevOps

The default configuration of the source controller does not use a Git implementation compatible with Azure DevOps.

To resolve this, you must configure the source-template's parameter `gitImplementation` to `libgit2`. Use one of these options:

- Use the tap-value `git_implementation` to set the parameter for the supply chain. See source-provider.
- Use the workload parameter `gitImplementation` to configure the parameter for the individual workload. See Parameters.

If both methods are set and do not match, the workload's parameter is respected.

Using Azure DevOps as a GitOps repository

The supply chain commits Kubernetes configuration to a Git repository. This configuration is then applied to another cluster. This is the GitOps promotion pattern.

You must construct a path and configure your Git implementation to read and write to an Azure DevOps repository.

GitOps write path example

The following example uses the Azure DevOps Git repository:

`https://dev.azure.com/vmware-tanzu/tap/_git/tap`

Set the `gitops_server_kind` workload parameters to `azure`.

```
ootb_supply_chain_testing_scanning:
  gitops:
    server_address: https://dev.azure.com
    repository_owner: vmware-tanzu/tap
    repository_name: tap
    pull_request:
      server_kind: azure
```

or the workload parameters:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  ...
spec:
  params:
    - name: gitops_server_address
      value: https://dev.azure.com
    - name: gitops_repository_owner
      value: vmware-tanzu/tap
    - name: gitops_repository_name
      value: tap
    - name: gitops_server_kind
      value: azure
    ...
  spec:
    params:
      - name: gitops_server_kind
        value: azure
      ...
```

Set other GitOps values in either tap-values or in the workload parameters.

- By using tap-values:

  ```
  ootb_supply_chain_testing_scanning:
    gitops:
      server_address: https://dev.azure.com
      repository_owner: vmware-tanzu/tap
      repository_name: tap
  ```

- By using the workload parameters:

  ```
  apiVersion: carto.run/v1alpha1
  kind: Workload
  metadata:
    ...
  spec:
    params:
      - name: gitops_server_address
        value: https://dev.azure.com
      - name: gitops_repository_owner
        value: vmware-tanzu/tap
      - name: gitops_repository_name
        value: tap
      ...
  ```

Gitops write path templates

Azure DevOps and Git use different URL structures.

For example, the Git clone URL of an Azure DevOps repository is structured as:

`https://dev.azure.com/<org_name>/<project_name>/_git/<repository_name>`

GitHub uses the following address structure:

`https://github.com/<org_name>/<repository_name>`

In Azure DevOps, a project can have multiple repositories, but the project name and repository name are often the same.

The config-writer and config-writer-and-pull-requester templates accept three parameters to build the path of the repository. For Azure DevOps, configure them as follows:

- gitops_server_address: `https://dev.azure.com`
- gitops_repository_owner: `<org_name>/<project_name>`
- gitops_repository_name: `<repository_name>`

Configure the template parameters as follows:

- `gitops.server_address` tap-value during the Out of the Box Supply Chains package installation or `gitops_server_address` configured as a workload parameter.
- `gitops.repository_owner` tap-value during the Out of the Box Supply Chains package installation or `gitops_repository_owner` configured as a workload parameter.
- `gitops.repository_name` tap-value during the Out of the Box Supply Chains package installation or `gitops_repository_name` configured as a workload parameter.

To properly contruct the write path, the template parameter `gitops_server_kind` must be configured as `azure`.

- Use the `gitops.pull_request.server_kind` tap-value during the Out of the Box Supply Chains package installation
- or configure`gitops_server_kind` as a workload parameter

Note

Even if the commit strategy is not pull-request, such as direct commits, to use an Azure DevOps repository either the tap value `gitops.pull_request.server_kind` or the workload parameter `gitops_server_kind` must be configured to `azure`.

When you use pull requests with GitOps, you can set the type of server with the tap-value `gitops.pull_request.server_kind`. See GitOps versus RegistryOps.

For information about configuring the GitOps write operations, see GitOps versus RegistryOps.

Gitops read example

The following example uses the Azure DevOps GitOps repository:

`https://dev.azure.com/vmware-tanzu/tap/_git/tap`

You can configure the delivery tap-values:

```
ootb_delivery_basic:
  git_implementation: libgit2
```

or the deliverable parameter:

```
apiVersion: carto.run/v1alpha1
kind: Deliverable
```

```
metadata:
  ...
spec:
  params:
    - name: gitImplementation
      value: libgit2
```

Gitops read implementation templates

Similar to reading an Azure DevOps source repo, when reading an AzureDevOps GitOps repository, you must configure the Git implementation for the delivery-source-template. This parameter's configuration comes from the delivery or the deliverable.

You can configure the delivery by using tap-values.

The supply chain creates the definition of a deliverable. Tanzu Application Platform users are responsible for applying this definition to the run cluster. Users can choose to add the `gitImplementation` parameter to the deliverable.

Author your supply chains

The Out of the Box Supply Chain, Delivery Basic, and Templates Supply Chain Choreographer packages give you Kubernetes objects that cover a reference path to production. Because VMware recognizes that you have your own needs, these objects are customizable, including individual templates for each resource, whole supply chains, or delivery objects.

Depending on how you installed Tanzu Application Platform, there are different ways to customize the Out of the Box Supply Chains. The following sections describe the ways supply chains and templates are authored within the context of profile-based Tanzu Application Platform installations.

Providing your own supply chain

To create a new supply chain and make it available for workloads, ensure that the supply chain does not conflict with those installed on the cluster, as those objects are cluster-scoped.

If this is your first time creating a supply chain, follow the tutorials from the Cartographer documentation.

Any supply chain installed in a Tanzu Application Platform cluster might encounter two possible cases of collisions:

- **object name**: Supply chains are cluster scoped, such as any Cartographer resource prefixed with `Cluster`. So the name of the custom supply chain must be different from the ones provided by the Out of the Box packages.

  Either create a supply chain whose name is different, or remove the installation of the corresponding `ootb-supply-chain-*` from the Tanzu Application Platform.

- **workload selection**: A workload is reconciled against a particular supply chain based on a set of selection rules as defined by the supply chains. If the rules for the supply chain to match a workload are ambiguous, the workload does not make any progress.

  Either create a supply chain whose selection rules are different from the ones the Out of the Box Supply Chain packages use, or remove the installation of the corresponding `ootb-supply-chain-*` from Tanzu Application Platform.

  See Selectors.

For Tanzu Application Platform 1.2, the following selection rules are in place for the supply chains of the corresponding packages:

- *ootb-supply-chain-basic*
    - ClusterSupplyChain/**basic-image-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
        - `workload.spec.image` text box set
    - ClusterSupplyChain/**source-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
    - ClusterSupplyChain/**basic-image-to-url-package (experimental)**
        - label `apps.tanzu.vmware.com/workload-type: server`
        - label `apps.tanzu.vmware.com/carvel-package-workflow: true`
    - ClusterSupplyChain/**source-to-url-package (experimental)**
        - label `apps.tanzu.vmware.com/workload-type: server`
        - label `apps.tanzu.vmware.com/carvel-package-workflow: true`
- *ootb-supply-chain-testing*
    - ClusterSupplyChain/**testing-image-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
        - `workload.spec.image` text box set
    - ClusterSupplyChain/**source-test-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
        - label `apps.tanzu.vmware.com/has-test: true`
- *ootb-supply-chain-testing-scanning*
    - ClusterSupplyChain/**scanning-image-scan-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
        - `workload.spec.image` text box set
    - ClusterSupplyChain/**source-test-scan-to-url**
        - label `apps.tanzu.vmware.com/workload-type: web`
        - label `apps.tanzu.vmware.com/has-test: true`

For details about how to edit an existing supply chain, see Modifying an Out of the Box Supply Chain section.

You can exclude a supply chain package from the installation to prevent the conflicts mentioned earlier, by using the `excluded_packages` property in `tap-values.yaml`. For example:

```
# add to exclued_packages `ootb-*` packages you DON'T want to install
# excluded_packages:
  - ootb-supply-chain-basic.apps.tanzu.vmware.com
  - ootb-supply-chain-testing.apps.tanzu.vmware.com
  - ootb-supply-chain-testing-scanning.apps.tanzu.vmware.com
# comment out remove the `supply_chain` property
#
# supply_chain: ""
```

Providing your own templates

Similar to supply chains, Cartographer templates (`Cluster*Template` resources) are cluster-scoped, so you must ensure that the new templates submitted to the cluster do not conflict with those installed by the `ootb-templates` package.

The following set of objects are provided by `ootb-templates`:

- ClusterConfigTemplate/**config-template**
- ClusterConfigTemplate/**convention-template**
- ClusterDeploymentTemplate/**app-deploy**
- ClusterImageTemplate/**image-provider-template**
- ClusterImageTemplate/**image-scanner-template**
- ClusterImageTemplate/**kpack-template**
- ClusterTask/**kaniko-build**
- ClusterImageTemplate/**kaniko-template**
- ClusterRole/**ootb-templates-app-viewer**
- ClusterRole/**ootb-templates-deliverable**
- ClusterRole/**ootb-templates-workload**
- ClusterRunTemplate/**tekton-source-pipelinerun**
- ClusterRunTemplate/**tekton-taskrun**
- ClusterSourceTemplate/**delivery-source-template**
- ClusterSourceTemplate/**source-scanner-template**
- ClusterSourceTemplate/**source-template**
- ClusterSourceTemplate/**testing-pipeline**
- ClusterTask/**git-writer**
- ClusterTask/**image-writer**
- ClusterTemplate/**config-writer-template**
- ClusterTemplate/**deliverable-template**
- ClusterTask/**carvel-package (experimental)**
- ClusterConfigTemplate/**carvel-package (experimental)**
- ClusterTemplate/**package-config-writer-and-pull-requester-template (experimental)**
- ClusterTemplate/**package-config-writer-template (experimental)**

Before submitting your own, either ensure that the name and resource has no conflicts with those installed by `ootb-templates`, or exclude from the installation the template you want to override by using the `excluded_templates` property of `ootb-templates`.

For example, perhaps you want to override the `ClusterConfigTemplate` named `config-template` to provide your own with the same name, so that you don't need to edit the supply chain. In `tap-values.yaml`, you can exclude template provided by Tanzu Application Platform:

```
ootb_templates:
  excluded_templates:
    - 'config-writer'
```

For details about how to edit an existing template, see Modifying an Out of the Box Supply template section.

Modifying an Out of the Box Supply Chain

To change the shape of a supply chain or the template that it points to, do the following:

1. Copy one of the reference supply chains.
2. Remove the old supply chain. See preventing Tanzu Application Platform supply chains from being installed.
3. Edit the supply chain object.
4. Submit the modified supply chain to the cluster.

Example

In this example, you have a new `ClusterImageTemplate` object named `foo` that you want use for building container images instead of the out of the box object that makes use of Kpack. The supply chain that you want to apply the modification to is `source-to-url` provided by the `ootb-supply-chain-basic` package.

1. Find the image that contains the supply chain definition:

```
kubectl get app ootb-supply-chain-basic \
  -n tap-install \
  -o jsonpath={.spec.fetch[0].imgpkgBundle.image}
```

```
registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:f2ad40
1bb3e850940...
```

2. Pull the contents of the bundle into a directory named `ootb-supply-chain-basic`:

```
imgpkg pull \
  -b registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:f
2ad401bb3e850940... \
  -o ootb-supply-chain-basic
```

```
Pulling bundle 'registry.tanzu.vmware.com/tanzu-...
  Extracting layer 'sha256:542f2bb8eb946fe9d2c8a...

Locating image lock file images...
The bundle repo (registry.tanzu.vmware.com/tanzu...

Succeeded
```

3. Inspect the files obtained:

```
tree ./ootb-supply-chain-basic/
```

```
./ootb-supply-chain-basic/
├── config
│   ├── supply-chain-image.yaml
│   └── supply-chain.yaml
└── values.yaml
```

4. Edit the supply chain that you want to exchange the template with another:

```
--- a/supply-chain.yaml
+++ b/supply-chain.yaml
@@ -52,7 +52,7 @@ spec:
   - name: image-builder
     templateRef:
       kind: ClusterImageTemplate
-      name: kpack-template
+      name: foo
     params:
```

```
      - name: serviceAccount
        value: #@ data.values.service_account
```

5. Submit the supply chain to Kubernetes:

   The supply chain definition found in the bundle expects the values you provided by using `tap-values.yaml` to be interpolated by using YTT before they are submitted to Kubernetes. So before applying the modified supply chain to the cluster, use YTT to interpolate those values. After that, run:

   ```
   ytt \
     --ignore-unknown-comments \
     --file ./ootb-supply-chain-basic/config \
     --data-value registry.server=REGISTRY-SERVER \
     --data-value registry.repository=REGISTRY-REPOSITORY |
     kubectl apply -f-
   ```

   Important

   The modified supply chain does not outlive the destruction of the cluster. VMware recommends that you save it, for example, in a Git repository to install on every cluster where you expect the supply chain to exist.

Modifying an Out of the Box Supply template

The Out of the Box Templates package (`ootb-templates`) includes all of the templates and shared Tekton tasks used by the supply chains shipped by using `ootb-supply-chain-*` packages. Any template that you want to edit, for example, to change details about the resources that are created based on them, is part of this package.

The workflow for updating a template is as follows:

1. Copy one of the reference templates from `ootb-templates`.
2. Exclude that template from the set of objects provided by `ootb-templates`. For more information, see `excluded_templates` in Providing your Own Templates.
3. Edit the template.
4. Submit the modified template to the cluster.

Note

You don't need to change anything related to supply chains, because you're preserving the name of the object referenced by the supply chain.

Example

In this example, you want to update the `ClusterImageTemplate` object called `kpack-template`, which provides a template for creating `kpack/Image`s to hardcode an environment variable.

1. Exclude the `kpack-template` from the set of templates that `ootb-templates` installs by upating `tap-values.yaml`:

   ```
   ootb_templates:
   excluded_templates: ['kpack-template']
   ```

2. Find the image that contains the templates:

```
kubectl get app ootb-templates \
  -n tap-install \
  -o jsonpath={.spec.fetch[0].imgpkgBundle.image}
```

```
registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a5e177
f38d7287f2ca7ee2afd67ff178645d8f1b1e47af4f192a5ddd6404825e
```

3. Pull the contents of the bundle into a directory named `ootb-templates`:

```
imgpkg pull \
  -b registry.tanzu.vmware.com/tanzu-application-platform/tap-packages@sha256:a
5e177f38d7.. \
  -o ootb-templates
```

```
Pulling bundle 'registry.tanzu.vmware.com/tanzu-...
  Extracting layer 'sha256:a5e177f38d7...

Locating image lock file images...
The bundle repo (registry.tanzu.vmware.com/tanzu...

Succeeded
```

4. Confirm that you downloaded all the templates:

```
tree ./ootb-templates
```

```
./ootb-templates
├── config
│   ├── cluster-roles.yaml
│   ├── config-template.yaml
│   ├── kpack-template.yaml         # ! the one we want to modify
...
│   └── testing-pipeline.yaml
└── values.yaml
```

5. Change the property you want to change:

```
--- a/config/kpack-template.yaml
+++ b/config/kpack-template.yaml
@@ -65,6 +65,8 @@ spec:
         subPath: #@ data.values.workload.spec.source.subPath
       build:
         env:
+        - name: FOO
+          value: BAR
         - name: BP_OCI_SOURCE
           value: #@ data.values.source.revision
         #@ if/end param("live-update"):
```

6. Submit the template.

The name of the template is preserved but the contents are changed. So after the template is submitted, the supply chains are all embedded to the build of the application container images that have `FOO` environment variable.

Live modification of supply chains and templates

Preceding sections covered how to update supply chains or templates installed in a cluster. This section shows how you can experiment by making small changes in a live setup with `kubectl edit`.

When you install Tanzu Application Platform by using profiles, a `PackageInstall` object is created. This in turn creates a set of children `PackageInstall` objects for installing the individual components that make up the platform.

```
PackageInstall/tap
└─App/tap
  ├─ PackageInstall/cert-manager
  ├─ PackageInstall/cartographer
  ├─ ...
  └─ PackageInstall/tekton-pipelines
```

Because the installation is based on Kubernetes primitives, `PackageInstall` tries to achieve the state where all packages are installed.

This is great but presents challenges for modifying the contents of some of the objects that the installation submits to the cluster. Namely, such modifications cause the original definition persisting instead of the changes.

For this reason, before you perform any customization to the Out of the Box packages, you must pause the top-level `PackageInstall/tap` object. Run:

```
kubectl edit -n tap-install packageinstall tap
```

```
apiVersion: packaging.carvel.dev/v1alpha1
kind: PackageInstall
metadata:
  name: tap
  namespace: tap-install
spec:
  paused: true                    # ! set this field to `paused: true`.
  packageRef:
    refName: tap.tanzu.vmware.com
    versionSelection:
# ...
```

With the installation of Tanzu Application Platform paused, all of the installed components are still there, but changes to those children `PackageInstall` objects are not overwritten.

Now you can pause the `PackageInstall` objects that relate to the templates or supply chains you want to edit.

For example:

- To edit templates: `kubectl edit -n tap-install packageinstall ootb-templates`
- To edit the basic supply chains: `kubectl edit -n tap-install packageinstall ootb-supply-chain-basic`

setting `packageinstall.spec.paused: true`.

With the installations paused, further live changes to templates or supply chains are persisted until you revert the `PackageInstall`s to not being paused. To persist the changes, follow the steps outlined in the earlier sections.

Adding custom behavior to Supply Chains

Most behaviors in supply chains are supplied by Kubernetes controllers. For example, Cloud Native Buildpacks are created by the kpack controller when a kpack Image object is created. Sometimes there is need for behavior and no controller for it exists. In these instances, you might want to write a script that uses a CLI tool, or interact with an external API. To do this, you can bring the behavior to the supply chain by using Tekton.

You can look at the kaniko image-building as an example. You create a Tekton ClusterTask `kaniko-build` with instructions for how to build a Docker image using kaniko given a set of parameters. The ClusterTask has a set of steps. Each step refers to a container image and a set of instructions to run on the image. For example, it can be a Linux image against which a set of bash instructions are run. The ClusterTask is installed on the cluster.

You create the ClusterImageTemplate `kaniko-template` to create Tekton TaskRuns. TaskRuns are immutable, so you add the `lifecycle: tekton` field to the template's specifications. This ensures two things:

- When inputs to the template change, rather than updating the TaskRun, a new TaskRun is created.
- Only the values from the most recently created TaskRun that is successful are propagated forward in the supply chain.

To learn more about the `lifecycle: tekton` field, see the Cartographer tutorial Lifecycle: Templating Objects That Cannot Update. To learn more about Tekton, see the Tekton documentation.

Reference guides for Supply Chain Choreographer for Tanzu

This topic describes the reference guides you can use for Supply Chain Choreographer for Tanzu.

Reference guides

The following reference guides apply to Supply Chain Choreographer for Tanzu:

- Tanzu Build Service Integration

Events reference for Supply Chain Choreographer

This topic describes each event you can view with Supply Chain Choreographer.

Events are emitted when Choreographer edits resources or notices a change in their output or healthy state. Don't treat events like logs, however they can offer valuable insight into what's happening in a supply chain over time. For example, very high occurrences of events in a short period of time might be a sign of slow application-level processing due to many page faults and a lack of storage resources.

Events are published on Workload, Deliverable, and Runnable resources. You can view them manually using:

```
kubectl describe workload.carto.run <workload-name> -n <workload-ns>
kubectl describe runnable.carto.run <runnable-name> -n <runnable-ns>
kubectl describe deliverable.carto.run <deliverable-name> -n <deliverable-ns>
```

Events

The following sections define the different events.

StampedObjectApplied

This event is emitted every time Choreographer creates or updates a resource. The created or updated resource is referenced in the event message.

Example messages:

```
Created object [gitrepositories.source.toolkit.fluxcd.io/my-project]
Updated object [apps.kappctrl.k14s.io/my-project-app]
```

StampedObjectRemoved

This event is emitted every time Choreographer deletes a resource. This currently only occurs when Runnable resources expire. The deleted object is referenced in the event message.

Example message:

```
Deleted object [task.tekton.dev/my-project-a737bdf]
```

ResourceOutputChanged

This event is emitted every time Choreographer recognizes a new output from a resource.

Example message:

```
[source-provider] found a new output in [imagerepositories.source.apps.tanzu.vmware.co
m/app]
```

ResourceHealthyStatusChanged

This event is emitted every time Choreographer recognizes that the healthy status of a resource has changed.

Example message:

```
[image-provider] found healthy status in [images.kpack.io/app] changed to [True]
[source-provider] found healthy status in [[gitrepositories.source.toolkit.fluxcd.io/m
y-project]] changed to [False]
```

Workload Reference for Supply Chain Choreographer

This topic describes the fields you can use for Supply Chain Choreographer workloads.

Standard Fields

Cartographer workloads have standard fields leveraged by supply chains. See Cartographer's Reference Documentation in the Cartographer documentation.

Labels

Workload labels affect which supply chain is selected. For information about which template is defined for a particular reference, see Selectors in the Cartographer documentation. Individual templates can also use workload labels.

The following are workload label keys whose values change the behavior of OOTB Supply Chains:

- `apps.tanzu.vmware.com/has-tests` by Source-Test-to-URL and Source-Test-Scan-to-URL.
- `apps.tanzu.vmware.com/workload-type` by all supply chains.
- `apis.apps.tanzu.vmware.com/register-api` by the Api-Descriptors Template.
- `apps.tanzu.vmware.com/carvel-package-workflow` by source-to-url-package (experimental) and basic-image-to-url-package (experimental).

Parameters

The OOTB templates are configured with parameters from the supply chain or workload. For information about Cartographer parameters, including precedence rules, see Parameters in the Cartographer documentation.

What parameters are relevant depends on the supply chain that selects the workload, for two reasons:

1. The OOTB supply chains refer to overlapping sets of templates. A workload selected by the Source-to-URL supply chain can provide a `scanning_image_template` parameter, but the supply chain does not refer to a template that leverages that parameter.
2. You can write Supply Chains to provide a parameter value to a template and prevent the workload from overriding the value. See Further Information in the Cartographer documentation.

The following list of parameters are respected by some OOTB supply chains. Each provides the templates that respect the parameter. The reference for the template details which supply chains include the template.

- gitImplementation: source-template
- gitops_ssh_secret: source-template, deliverable-template, external-deliverable-template
- serviceAccount: source-template, image-provider-template, kpack-template, kaniko-template, convention-template, config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- maven: source-template
- testing_pipeline_matching_labels: testing-pipeline
- testing_pipeline_params: testing-pipeline
- scanning_source_template: source-scanner-template
- scanning_source_policy: source-scanner-template
- clusterBuilder: kpack-template
- buildServiceBindings: kpack-template
- live-update: kpack-template, convention-template
- dockerfile: kaniko-template
- docker_build_context: kaniko-template
- docker_build_extra_args: kaniko-template
- scanning_image_template: image-scanner-template
- scanning_image_policy: image-scanner-template
- annotations: convention-template, service-bindings, api-descriptors
- debug: convention-template
- ports: server-template
- api-descriptors: api-descriptors
- gitops_branch: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops_user_name: config-writer-template, config-writer-and-pull-requester-template
- gitops_user_email: config-writer-template, config-writer-and-pull-requester-template

- gitops_commit_message: config-writer-template, config-writer-and-pull-requester-template
- gitops_repository: config-writer-template, deliverable-template, external-deliverable-template
- gitops_repository_prefix: config-writer-template, deliverable-template, external-deliverable-template
- gitops_server_address: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops_repository_owner: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops_repository_name: config-writer-template, config-writer-and-pull-requester-template, deliverable-template, external-deliverable-template
- gitops_commit_branch: config-writer-and-pull-requester-template
- gitops_pull_request_title: config-writer-and-pull-requester-template
- gitops_pull_request_body: config-writer-and-pull-requester-template
- gitops_server_kind: config-writer-and-pull-requester-template
- carvel_package_gitops_subpath (experimental): carvel-package, package-config-writer-template, package-config-writer-and-pull-requester-template
- carvel_package_name_suffix (experimental): carvel-package, package-config-writer-template, package-config-writer-and-pull-requester-template

Service Account

To create the templated objects, Cartographer needs a reference to a service account with permissions to manage resources. This service account might be provided in the workload's `.spec.serviceAccountName` field or in the supply chain's `spec.serviceAccountRef` field. See Service Account and Workload and Supply Chain Custom Resources in the Cartographer documentation. When using the Tanzu CLI to create a workload, specify this service account's name with the `--service-account` flag.

After the templated objects are created, they often need a service account with permissions to do work. In the OOTB Templates and Supply Chains, the parameter `serviceAccount` must reference the service account for these objects. When using the Tanzu CLI to create a workload, specify this service account's name with `--param serviceAccount=...`.

Supply chains for Supply Chain Choreographer

This topic describes the parameters for supply chains that you can use with Supply Chain Choreographer.

Tanzu Application Platform includes a number of supply chains packages, each of which installs two ClusterSupplyChains. You can only install one supply chain package at a time.

The supply chains provide some parameters to the referenced templates. The parameters provided by the workload might override the parameters in this topic.

Source-to-URL

Purpose

- Fetches application source code
- Builds it into an image
- Writes the Kubernetes configuration necessary to deploy the application
- Commits that configuration to either a Git repository or a container image registry

Resources

This section describes the templates and their parameters.

source-provider

Refers to source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `gitImplementation` from tap-value `git_implementation`. NOT overridable by workload.

image-provider

Refers to kaniko-template when the workload provides a parameter `dockerfile`. Refers to kpack-template otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Basic

More information

See Install Out of the Box Supply Chain Basic for information about setting tap-values at installation time.

Source-Test-to-URL

- Fetches application source code
- Runs user defined tests against the code
- Builds the code into an image
- Writes the Kubernetes configuration necessary to deploy the application
- Commits that configuration to either a Git repository or a container image registry

Resources

source-provider

Refers to source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `gitImplementation` from tap-value `git_implementation`. NOT overridable by workload.

source-tester

Refers to testing-pipeline.

No parameters are provided by the supply-chain.

image-provider

Refers to kaniko-template when the workload provides a parameter `dockerfile`. Refers to kpack-template otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer

- Deliverable

Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- See Params provided by all Supply Chains to all Resources.

Package

Out of the Box Supply Chain Testing

More information

See Install Out of the Box Supply Chain with Testing for information about setting tap-values at installation time.

Source-Test-Scan-to-URL

- Fetches application source code
- Runs user defined tests against the code
- Scans the code for vulnerabilities
- Builds the code into an image
- Scans the image for vulnerabilities
- Writes the Kubernetes configuration necessary to deploy the application
- Commits that configuration to either a Git repository or an image registry

Resources

source-provider

Refers to source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `gitImplementation` from tap-value `git_implementation`. NOT overridable by workload.

source-tester

Refers to testing-pipeline.

No parameters are provided by the supply-chain.

source-scanner

Refers to source-scanner-template.

Parameters provided:

- `scanning_source_policy` from tap-value `scanning.source.policy`. Overridable by workload.
- `scanning_source_template` from tap-value `scanning.source.template`. Overridable by workload.

image-provider

Refers to kaniko-template when the workload provides a parameter `dockerfile`. Refers to kpack-template otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

image-scanner

Refers to image-scanner-template.

Parameters provided:

- `scanning_image_policy` from tap-value `scanning.image.policy`. Overridable by workload.
- `scanning_image_template` from tap-value `scanning.image.template`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Testing Scanning

More information

See Install Out of the Box Supply Chain with Testing and Scanning for information about setting tap-values at installation time.

Basic-Image-to-URL

- Fetches a prebuilt image.

- Writes the Kubernetes configuration necessary to deploy the application.
- Commits that configuration to either a Git repository or an image registry.

Resources

image-provider

Refers to image-provider-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Basic

More information

See Install Out of the Box Supply Chain Basic for information about setting tap-values at installation time.

Testing-Image-to-URL

- Fetches a prebuilt image.
- Writes the Kubernetes configuration necessary to deploy the application.
- Commits that configuration to either a Git repository or an image registry.

Resources

image-provider

Refers to image-provider-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

Common resources

- Config-Provider

- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Testing

More information

See Install Out of the Box Supply Chain with Testing for information about setting tap-values at installation time.

Scanning-image-scan-to-URL

- Fetches a prebuilt image.
- Scans the image for vulnerabilities.
- Writes the Kubernetes configuration necessary to deploy the application.
- Commits the configuration to either a Git repository or an image registry.

Resources

image-provider

Refers to image-provider-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

image-scanner

Refers to image-scanner-template.

Parameters provided:

- `scanning_image_policy` from tap-value `scanning.image.policy`. Overridable by workload.
- `scanning_image_template` from tap-value `scanning.image.template`. Overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors

- Config-Writer
- Deliverable

Parameters provided to all resources

- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Testing Scanning

More information

See Install Out of the Box Supply Chain with Testing and Scanning for information about setting tap-values at installation time.

Source-to-URL-Package (experimental)

Purpose

- Fetches the application source code.
- Builds the source code into an image.
- Bundles the Kubernetes configuration necessary to deploy the application into a Carvel Package.
- Commits the Package to a Git Repository.

Resources

This section describes the templates and their parameters.

source-provider

Refers to source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `gitImplementation` from tap-value `git_implementation`. NOT overridable by workload.

image-provider

Refers to kaniko-template when the workload provides a parameter `dockerfile`. Refers to kpack-template otherwise.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.
- `clusterBuilder` from tap-value `cluster_builder`. Overridable by workload.
- `dockerfile` value `./Dockerfile`. Overridable by workload.
- `docker_build_context` value `./`. Overridable by workload.
- `docker_build_extra_args` value `[]`. Overridable by workload.

carvel-package

Refers to carvel-package.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

package-config-writer

Refers to the package-config-writer-and-pull-requester-template when the tap-value `gitops.commit_strategy` is `pull_request`. Otherwise, this resource refers to the package-config-writer-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors

Parameters provided to all resources

- `maven_repository_url` from tap-value `maven.repository.url`. NOT overridable by workload.
- `maven_repository_secret_name` from tap-value `maven.repository.secret_name`. NOT overridable by workload.
- `carvel_package_gitops_subpath` from tap-value `carvel_package.gitops_subpath`. Overridable by workload.
- `carvel_package_name_suffix` from tap-value `carvel_package.name_suffix`. Overridable by workload.
- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Basic

More information

See Install Out of the Box Supply Chain Basic for information about setting tap-values at installation time.

Basic-Image-to-URL-Package (experimental)

- Fetches a prebuilt image.
- Bundles the Kubernetes configuration necessary to deploy the application into a Carvel Package.
- Commits the Package to a Git Repository.

Resources

image-provider

Refers to image-provider-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

carvel-package

Refers to carvel-package.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

package-config-writer

Refers to the package-config-writer-and-pull-requester-template when the tap-value `gitops.commit_strategy` is `pull_request`. Otherwise, this resource refers to the package-config-writer-template

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

Common resources

- Config-Provider
- App-Config
- Service-Bindings
- Api-Descriptors
- Config-Writer
- Deliverable

Parameters provided to all resources

- `carvel_package_gitops_subpath` from tap-value `carvel_package.gitops_subpath`. Overridable by workload.
- `carvel_package_name_suffix` from tap-value `carvel_package.name_suffix`. Overridable by workload.
- See Params provided by all Supply Chains to all Resources

Package

Out of the Box Supply Chain Basic

More information

See Install Out of the Box Supply Chain Basic for information about setting tap-values at installation time.

Resources common to all OOTB supply chains

config-provider

Refers to convention-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.

app-config

The tap-values field `supported_workloads` defines which templates are referred to by this resource. Default configuration is:

```
supported_workloads:
- type: web
  cluster_config_template_name: config-template
- type: server
  cluster_config_template_name: server-template
- type: worker
  cluster_config_template_name: worker-template
```

The workload's `apps.tanzu.vmware.com/workload-type` label determines which template is used at this step. For example, when the workload has a label `apps.tanzu.vmware.com/workload-type:web`, the supply chain references `config-template`.

No parameters are provided by the supply-chain.

service-bindings

Refers to the service-binding template.

No parameters are provided by the supply-chain.

api-descriptors

Refers to the api-descriptors template.

No parameters are provided by the supply-chain.

config-writer

Refers to the config-writer-and-pull-requester-template when the tap-value `gitops.commit_strategy` is `pull_request`. Otherwise, this resource refers to the config-writer-template

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by workload.
- `registry` from tap-value `registry`. NOT overridable by workload.

deliverable

Refers to the external-deliverable-template when the tap-value `external_delivery` evaluates to `true`. Otherwise the resource refers to the deliverable-template.

Parameters provided:

- `registry` from tap-value `registry`. NOT overridable by workload.

Parameters provided by all supply chains to all resources

All of the following parameters are overridable by the workload.

- `gitops_branch` from tap-value `gitops.branch`
- `gitops_user_name` from tap-value `gitops.username`
- `gitops_user_email` from tap-value `gitops.email`
- `gitops_commit_message` from tap-value `gitops.commit_message`
- `gitops_ssh_secret` from tap-value `gitops.ssh_secret`
- `gitops_repository_prefix` from tap-value `gitops.repository_prefix` when present.
- `gitops_server_address` from tap-value `gitops.server_address` when present.
- `gitops_repository_owner` from tap-value `gitops.repository_owner` when present.
- `gitops_repository_name` from tap-value `gitops.repository_name` when present.
- `gitops_server_kind` from tap-value `gitops.pull_request.server_kind` when present.
- `gitops_commit_branch` from tap-value `gitops.pull_request.commit_branch` when present.
- `gitops_pull_request_title` from tap-value `gitops.pull_request.pull_request_title` when present.
- `gitops_pull_request_body` from tap-value `gitops.pull_request.pull_request_body` when present.

Template reference for Supply Chain Choreographer

This topic describes the objects from templates that you can use with Supply Chain Choreographer.

All the objects referenced in this topic are Cartographer Templates packaged in Out of the Box Templates.

This topic describes:

- The purpose of the templates
- The one or more objects that the templates create
- The supply chains that include the templates
- The parameters that the templates use

source-template

Purpose

Creates an object to fetch source code and make that code available to other objects in the supply chain. See Building from Source.

Used by

- Source-to-URL in the `source-provider` step.
- Source-Test-to-URL in the `source-provider` step.
- Source-Test-Scan-to-URL in the `source-provider` step.
- Source-to-URL-Package (experimental) in the `source-provider` step.

Creates

The source-template creates one of three objects, either:

- GitRepository. Created if the workload has `.spec.source.git` defined.

- MavenArtifact. Created if the template is provided a value for the parameter `maven`.
- ImageRepository. Created if the workload has `.spec.source.image` defined.

GitRepository

`GitRepository` makes source code from a particular commit available as a tarball in the cluster. Other resources in the supply chain can then access that code.

Parameters

Template reference for Supply Chain Choreographer

Parameter name	Meaning	Example
`gitImplementation`	The library used to fetch source code. If not provided, Tanzu Application Platform's default implementation uses `go-git`, which works with the providers supported by Tanzu Application Platform: GitHub and GitLab. An alternate value that can be used with other Git providers is `libgit2`.	`- name: gitImplementation` `value: libgit2`
`gitops_ssh_secret`	Name of the secret used to provide credentials for the Git repository. The secret with this name must exist in the same namespace as the `Workload`. The credentials must be sufficient to read the repository. If not provided, Tanzu Application Platform defaults to look for a secret named `git-ssh`. See Git authentication.	`- name: gitops_ssh_secret` `value: git-credentials`

Note

Some Git providers, notably Azure DevOps, require you to use `libgit2` due to the server-side implementation providing support only for git's v2 protocol. For information about the features supported by each implementation, see git implementation in the flux documentation.

More information

For an example using the Tanzu CLI to create a Workload using GitHub as the provider of source code, see Create a workload from GitHub repository.

For information about GitRepository objects, see GitRepository.

ImageRepository

`ImageRepository` makes the contents of a container image available as a tarball on the cluster.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account, providing credentials to `ImageRepository` for fetching container images. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount value: default`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about the ImageRepository resource, see the ImageRepository reference documentation.

For information about how to use the Tanzu CLI to create a workload leveraging ImageRepository, see Create a workload from local source code.

MavenArtifact

`MavenArtifact` makes a pre-built Java artifact available to as a tarball on the cluster.

While the `source-template` leverages the workload's `.spec.source` field when creating a `GitRepository` or `ImageRepository` object, the creation of the `MavenArtifact` relies only on parameters in the Workload.

Parameters

Parameter name	Meaning	Example
`maven`	Points to the Maven artifact to fetch and the polling interval.	`- name: maven value: artifactId: springboot-initial groupId: com.example version: RELEASE classifier: sources # optional type: jar # optional artifactRetryTimeout: 1m0s # optional`

Parameter name	Meaning	Example
`maven_repository_url`	Specifies the Maven repository from which to fetch	`- name: maven_repository_url` `value: https://repo1.maven.org/maven2/`
`maven_repository_secret_name`	Specifies the secret containing credentials necessary to fetch from the Maven repository. The secret named must exist in the same workspace as the workload.	`- name: maven_repository_secret_name` `value: auth-secret`

More information

For information about the custom resource, see MavenArtifact reference docs.

For information about how to use the custom resource with the `tanzu apps workload` CLI plug-in Create a Workload from Maven repository artifact.

testing-pipeline

Purpose

Tests the source code provided in the supply chain. Testing depends on a user provided Tekton Pipeline. Parameters for this template allow for selection of the proper Pipeline and for specification of additional values to pass to the Pipeline.

Used by

- Source-Test-to-URL in the source-tester step.
- Source-Test-Scan-to-URL in the source-tester step.

These are used as the `source-tester` resource.

Creates

`testing-pipeline`creates a Runnable object. This Runnable provides inputs to the ClusterRunTemplate named tekton-source-pipelinerun.

Parameters

Parameter name	Meaning	Example
`testing_pipeline_matching_labels`	Set of labels to use when searching for Tekton Pipeline objects in the same namespace as the Workload. By default, a Pipeline labeled as `apps.tanzu.vmware.com/pipeline: test` is selected.	`- name: testing_pipeline_matching_labels value: apps.tanzu.vmware.com/pipeline: test my.company/language: golang`
`testing_pipeline_params`	Set of parameters to pass to the Tekton Pipeline. To this set of parameters, the template always adds the source URL and revision as `source-url` and `source-revision`.	`- name: testing_pipeline_params value: - name: verbose value: true - name: foo value: bar`

More information

For information about the ClusterRunTemplate that pairs with the Runnable, read tekton-source-pipelinerun

For information about the Tekton Pipeline that the user must create, read the OOTB Supply Chain Testing documentation of the Pipeline

source-scanner-template

Purpose

Scans the source code for vulnerabilities.

Used by

- Source-Test-Scan-to-URL in the source-scanner step.

This is used as the `source-scanner` resource.

Creates

SourceScan

Parameters

Parameter name	Meaning	Example
`scanning_source_template`	Name of the ScanTemplate object to use for running the scans. The ScanTemplate must be in the same namespace as the Workload.	`- name: scanning_source_template value: private-source-scan-template`
`scanning_source_policy`	Name of the ScanPolicy object to use when evaluating the scan results of a source scan. The ScanPolicy must be in the same namespace as the Workload.	`- name: scanning_source_policy value: allowlist-policy`

More information

For information about how to set up the Workload namespace with the ScanPolicy and ScanTemplate required for this resource, see Out of the Box Supply Chain with Testing and Scanning.

For information about the SourceScan custom resource, see SourceScan reference.

For information about how the artifacts found during scanning are catalogued, see Supply Chain Security Tools for Tanzu – Store.

image-provider-template

Purpose

Fetches a container image of a prebuilt application, specified in the workload's `.spec.image` field. This makes the content-addressable name, (e.g. the image name containing the digest) available to other resources in the supply chain.

Used by

- Basic-Image-to-URL in the image-provider step.
- Testing-Image-to-URL in the image-provider step.
- Scanning-Image-Scan-to-URL in the image-provider step.
- Basic-Image-to-URL-Package (experimental) in the image-provider step.

These are used as the `image-provider` resource.

Creates

ImageRepository.source.apps.tanzu.vmware.com

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing credentials for the target image registry. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount value: default`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about the ImageRepository resource, see ImageRepository reference docs.

For information about prebuilt images, see Using a prebuilt image.

kpack-template

Purpose

Builds an container image from source code using cloud native buildpacks.

Used by

- Source-to-URL in the image-provider step.
- Source-Test-to-URL in the image-provider step.
- Source-Test-Scan-to-URL in the image-provider step.
- Source-to-URL-Package (experimental) in the image-provider step.

These are used as the `image-provider` resource when the workload parameter `dockerfile` is not defined.

Creates

Image.kpack.io

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing credentials for the configured image registry. `Image` uses these credentials to push built container images to the registry. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount value: default`

Parameter name	Meaning	Example
`clusterBuilder`	Name of the Kpack Cluster Builder to use.	`- name: clusterBuilder value: nodejs-cluster-builder`
`buildServiceBindings`	Definition of a list of service bindings to make use at build time. For example, providing credentials for fetching dependencies from repositories that require credentials.	`- name: buildServiceBindings value: - name: settings-xml kind: Secret apiVersion: v1`
`live-update`	Enable the use of Tilt's live-update function.	`- name: live-update value: "true"`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about the integration with Tanzu Build Service, see Tanzu Build Service Integration.

For information about `live-update`, see Developer Conventions and Overview of Tanzu Developer Tools for IntelliJ.

For information about using Kpack builders with `clusterBuilder`, see Builders.

For information about `buildServiceBindings`, see Service Bindings.

kaniko-template

Purpose

Build an image for source code that includes a Dockerfile.

Used by

- Source-to-URL in the image-provider step.
- Source-Test-to-URL in the image-provider step.

- Source-Test-Scan-to-URL in the image-provider step.
- Source-to-URL-Package (experimental) in the image-provider step.

These are used as the `image-provider` resource when the workload parameter `dockerfile` is defined.

Creates

A taskrun.tekton.dev which provides configuration to a Tekton ClusterTask to build an image with kaniko.

This template uses the lifecycle: tekton flag to create new immutable objects rather than updating the previous object.

Parameters

Parameter name	Meaning	Example
`dockerfile`	relative path to the Dockerfile file in the build context	`./Dockerfile`
`docker_build_context`	relative path to the directory where the build context is	`.`
`docker_build_extra_args`	List of flags to pass directly to kaniko,such as providing arguments to a build.	`- --build-arg=FOO=BAR`
`serviceAccount`	Name of the service account to use for providing Docker credentials. The service account must exist in the same namespace as the Workload. The service account must have a secret associated with the credentials. See Configuring authentication for Docker in the Tekton documentation.	`- name: serviceAccount value: default`
`registry`	Specification of the registry server and repository in which the built image is placed.	`- name: registry value: server: index.docker.io repository: web-team`

More information

For information about how to use Dockerfile-based builds and limits associated with the function, see Dockerfile-based builds.

For information about `lifecycle:tekton`, read Cartographer Lifecycle.

image-scanner-template

Purpose

Scans the container image for vulnerabilities, persists the results in a store, and prevents the image from moving forward if CVEs are found which are not compliant with its referenced ScanPolicy.

Used by

- Source-Test-Scan-to-URL in the image-scanner step.
- Scanning-Image-Scan-to-URL in the image-scanner step.

Creates

ImageScan.scanning.apps.tanzu.vmware.com

Parameters

Parameter name	Meaning	Example
`scanning_image_template`	Name of the ScanTemplate object for running the scans against a container image. The ScanTemplate must be in the same namespace as the Workload.	`- name: scanning_image_template value: private-image-scan-template`
`scanning_image_policy`	Name of the ScanPolicy object for evaluating the scan results of an image scan. The ScanPolicy must be in the same namespace as the Workload.	`- name: scanning_image_policy value: allowlist-policy`

More information

For information about the ImageScan custom resource, see ImageScan reference.

For information about how the artifacts found during scanning are catalogued, see Supply Chain Security Tools for Tanzu – Store.

convention-template

Purpose

Create the PodTemplateSpec for the Kubernetes configuration (e.g. the knative service or kubernetes deployment) which are applied to the cluster.

Used by

- Source-to-URL in the config-provider step.
- Basic-Image-to-URL in the config-provider step.
- Source-Test-to-URL in the config-provider step.
- Testing-Image-to-URL in the config-provider step.
- Source-Test-Scan-to-URL in the config-provider step.
- Scanning-Image-Scan-to-URL in the config-provider step.

- Source-to-URL-Package (experimental) in the config-provider step.
- Basic-Image-to-URL-Package (experimental) in the config-provider step.

Creates

Creates a PodIntent object. The PodIntent leverages conventions installed on the cluster. The PodIntent object is responsible for generating a PodTemplateSpec. The PodTemplateSpec is used in app configs, such as knative services and deployments, to represent the shape of the pods to run the application in containers.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the serviceAccount providing necessary credentials to `PodIntent`. The serviceAccount must be in the same namespace as the Workload. The serviceAccount is set as the `serviceAccountName` in the podtemplatespec. The credentials associated with the serviceAccount must allow fetching the container image used to inspect the metadata passed to convention servers.	`- name: serviceAccount` `value: default`

`annotations`	Extra set of annotations to pass down to the PodTemplateSpec.	`- name: annotations`
		`value:`
		`name: my-application`
		`version: v1.2.3`
		`team: store`

`debug`	Put the workload in debug mode.	`- name: debug` `value: "true"`
`live-update`	Enable live-updating of the code (for innerloop development).	`- name: live-update` `value: "true"`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about `PodTemplateSpec`, see PodTemplateSpec in the Kubernetes documentation.

For information about conventions, see Cartographer Conventions.

For information about the two convention servers enabled by default in Tanzu Application Platform installations, see Developer Conventions and Spring Boot conventions.

config-template

Purpose

For workloads with the label `apps.tanzu.vmware.com/workload-type: web`, define a knative service.

Used by

- Source-to-URL in the app-config step.
- Basic-Image-to-URL in the app-config step.
- Source-Test-to-URL in the app-config step.
- Testing-Image-to-URL in the app-config step.
- Source-Test-Scan-to-URL in the app-config step.
- Scanning-Image-Scan-to-URL in the app-config step.

Creates

A ConfigMap, in which the data field has a key `delivery.yaml` whose value is the definition of a knative service.

Parameters

None

More information

See workload types for more details about the three different types of workloads.

worker-template

Purpose

For workloads with the label `apps.tanzu.vmware.com/workload-type: worker`, define a Kubernetes Deployment.

Used by

- Source-to-URL in the app-config step.
- Basic-Image-to-URL in the app-config step.

- Source-Test-to-URL in the app-config step.
- Testing-Image-to-URL in the app-config step.
- Source-Test-Scan-to-URL in the app-config step.
- Scanning-Image-Scan-to-URL in the app-config step.

Creates

A ConfigMap, in which the data field has a key `delivery.yaml` whose value is the definition of a Kubernetes Deployment.

Parameters

None

More information

For information about the three different types of workloads, see workload types.

server-template

Purpose

For workloads with the label `apps.tanzu.vmware.com/workload-type: server`, define a Kubernetes Deployment and a Kubernetes Service.

Used by

- Source-to-URL in the app-config step.
- Basic-Image-to-URL in the app-config step.
- Source-Test-to-URL in the app-config step.
- Testing-Image-to-URL in the app-config step.
- Source-Test-Scan-to-URL in the app-config step.
- Scanning-Image-Scan-to-URL in the app-config step.
- Source-to-URL-Package (experimental) in the app-config step.
- Basic-Image-to-URL-Package (experimental) in the app-config step.

Creates

A ConfigMap, in which the data field has a key `delivery.yaml` whose value is the definitions of a Kubernetes Deployment and a Kubernetes Service to expose the pods.

Parameters

Parameter name	Meaning	Example

`ports`	Set of network ports to expose from the application to the Kubernetes cluster.	`- name: ports`
		`value:`
		`- containerPort: 2025`
		`name: smtp`
		`port: 25`

More information

For information about the three different types of workloads, see workload types.

For information about the ports parameter, see server-specific Workload parameters.

service-bindings

Purpose

Adds ServiceBindings to the set of Kubernetes configuration files.

Used by

- Source-to-URL in the service-bindings step.
- Basic-Image-to-URL in the service-bindings step.
- Source-Test-to-URL in the service-bindings step.
- Testing-Image-to-URL in the service-bindings step.
- Source-Test-Scan-to-URL in the service-bindings step.
- Scanning-Image-Scan-to-URL in the service-bindings step.
- Source-to-URL-Package (experimental) in the service-bindings step.
- Basic-Image-to-URL-Package (experimental) in the service-bindings step.

Creates

A ConfigMap. This template consumes input of multiple deployment YAML files and enriches the input with ResourceClaims and ServiceBindings if the workload contains serviceClaims.

Parameters

Parameter name	Meaning	Example

`annotations`	Extra set of annotations to pass down to the ServiceBinding and ResourceClaim objects.	`- name: annotations`
		`value:`
		`name: my-application`
		`version: v1.2.3`
		`team: store`

More information

For an example, see –service-ref in the Tanzu CLI documentation.

For an overview of the function, see Consume services on Tanzu Application Platform.

api-descriptors

Purpose

The `api-descriptor` resource takes care of adding an APIDescriptor to the set of Kubernetes objects to deploy such that API auto registration takes place.

Used by

- Source-to-URL in the api-descriptors step.
- Basic-Image-to-URL in the api-descriptors step.
- Source-Test-to-URL in the api-descriptors step.
- Testing-Image-to-URL in the api-descriptors step.
- Source-Test-Scan-to-URL in the api-descriptors step.
- Scanning-Image-Scan-to-URL in the api-descriptors step.
- Source-to-URL-Package (experimental) in the api-descriptors step.
- Basic-Image-to-URL-Package (experimental) in the api-descriptors step.

Creates

A ConfigMap. This template consumes input of multiple YAML files and enriches the input with an APIDescriptor if the workload has a label `apis.apps.tanzu.vmware.com/register-api` set to `true`.

Parameters

Parameter name	Meaning	Example

`annotations`	Extra set of annotations to pass down to the APIDescriptor object.	`- name: annotations`
		`value:`
		`name: my-application`
		`version: v1.2.3`
		`team: store`

`api_descriptor`	Information used to fill the state that you want of the APIDescriptor object (its spec).	`- name: api_descriptor`
		`value:`
		`type: openapi`
		`location:`
		`baseURL: http://petclinic-hard-coded.my-apps.tapdemo.vmware.com/`
		`path: "/v3/api`
		`owner: team-petclinic`
		`system: pet-clinics`
		`description: "example"`

More information

For information about API auto registration, see Use API Auto Registration.

config-writer-template

Purpose

Persist in an external system, such as a registry or git repository, the Kubernetes configuration passed to the template.

Used by

- Source-to-URL in the config-writer step.
- Basic-Image-to-URL in the config-writer step.
- Source-Test-to-URL in the config-writer step.
- Testing-Image-to-URL in the config-writer step.
- Source-Test-Scan-to-URL in the config-writer step.
- Scanning-Image-Scan-to-URL in the config-writer step.

Creates

A runnable which creates a Tekton TaskRun that refers either to the Tekton Task `git-writer` or the Tekton Task `image-writer`.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount` `value: default`
`gitops_branch`	Name of the branch to push the configuration to.	`- name: gitops_branch` `value: main`
`gitops_user_name`	User name to use in the commits.	`- name: gitops_user_name` `value: "Alice Lee"`
`gitops_user_email`	User email address to use in the commits.	`- name: gitops_user_email` `value: alice@example.com`

Parameter name	Meaning	Example
`gitops_commit_message`	Message to write as the body of the commits produced for pushing configuration to the Git repository.	`- name: gitops_commit_message` `value: "ci bump"`
`gitops_repository`	The full repository URL to which the configuration is committed. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/cartographer"`
`gitops_repository_prefix`	The prefix of the repository URL. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/"`
`gitops_server_address`	The server URL of the Git repository to which configuration is applied.	`- name: gitops_server_address` `value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`

`registry`	Specification of the registry server and repository in which the configuration is placed.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`
		`ca_cert_data:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

More information

For information about operating this template, see Gitops vs RegistryOps and the config-writer-and-pull-requester-template.

config-writer-and-pull-requester-template

Purpose

Persist the passed in Kubernetes configuration to a branch in a repository and open a pull request to another branch. This process allows for manual review of configuration before deployment to a cluster.

Used by

- Source-to-URL in the config-writer step.
- Basic-Image-to-URL in the config-writer step.
- Source-Test-to-URL in the config-writer step.
- Testing-Image-to-URL in the config-writer step.
- Source-Test-Scan-to-URL in the config-writer step.
- Scanning-Image-Scan-to-URL in the config-writer step.

Creates

A runnable which provides configuration to the ClusterRunTemplate `commit-and-pr-pipelinerun` to create a Tekton TaskRun. The Tekton TaskRun refers to the Tekton Task `commit-and-pr`.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount` `value: default`
`gitops_commit_branch`	Name of the branch to which configuration is pushed.	`- name: gitops_commit_branch` `value: feature`
`gitops_branch`	Name of the branch to which a pull request is opened.	`- name: gitops_branch` `value: main`

Parameter name	Meaning	Example
`gitops_user_name`	User name to use in the commits.	`- name: gitops_user_name` `value: "Alice Lee"`
`gitops_user_email`	User email address to use in the commits.	`- name: gitops_user_email` `value: alice@example.com`
`gitops_commit_message`	Message to write as the body of the commits produced for pushing configuration to the Git repository.	`- name: gitops_commit_message` `value: "ci bump"`
`gitops_pull_request_title`	Title of the pull request to be opened.	`- name: gitops_pull_request_title` `value: "ready for review"`
`gitops_pull_request_body`	Body of the pull request to be opened.	`- name: gitops_pull_request_body` `value: "generated by supply chain"`
`gitops_server_address`	The server URL of the Git repository to which configuration is applied.	`- name: gitops_server_address` `value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`

Parameter name	Meaning	Example
`gitops_server_kind`	The kind of Git provider	`- name: gitops_server_kind` `value: gitlab`

More information

For information about the operation of this template, see Gitops vs RegistryOps and the config-writer-template.

deliverable-template

Purpose

Create a deliverable which pairs with a Delivery to deploy Kubernetes configuration on the cluster.

Used by

- Source-to-URL in the deliverable step.
- Basic-Image-to-URL in the deliverable step.
- Source-Test-to-URL in the deliverable step.
- Testing-Image-to-URL in the deliverable step.
- Source-Test-Scan-to-URL in the deliverable step.
- Scanning-Image-Scan-to-URL in the deliverable step.

Creates

A Deliverable preconfigured with reference to a repository or registry from which to fetch Kubernetes configuration.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing the necessary permissions for the Delivery to create children objects. Populates the Deliverable's serviceAccount parameter. The service account must be in the same namespace as the Deliverable.	`- name: serviceAccount` `value: default`
`gitops_ssh_secret`	Name of the secret where credentials exist for fetching the configuration from a Git repository. Populates the Deliverable's gitops_ssh_secret parameter. The service account must be in the same namespace as the Deliverable.	`- name: gitops_ssh_secret` `value: ssh-secret`

Parameter name	Meaning	Example
`gitops_branch`	Name of the branch from which to fetch the configuration.	`- name: gitops_branch` `value: main`
`gitops_repository`	The full repository URL to which the configuration is fetched. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/cartographer"`
`gitops_repository_prefix`	The prefix of the repository URL. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/"`
`gitops_server_address`	The server URL of the Git repository from which configuration is fetched.	`- name: gitops_server_address` `value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`

Parameter name	Meaning	Example

`registry`	Specification of the registry server and repository from which the configuration is fetched.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`
		`ca_cert_data:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

Note

When using the Tanzu CLI to configure this `serviceAccount` parameter, use `--param serviceAccount=...`. The similarly named `--service-account` flag sets a different value: the `spec.serviceAccountName` key in the Workload object.

More information

For information about the ClusterDelivery shipped with `ootb-delivery-basic`, see Out of the Box Delivery Basic.

external-deliverable-template

Purpose

Create a definition of a deliverable which a user can manually applied to an external kubernetes cluster. When a properly configured Delivery is installed on that external cluster, the Deliverable will pair with the Delivery to deploy Kubernetes configuration on the cluster. For example, the OOTB Delivery.

Used by

- Source-to-URL in the deliverable step.
- Basic-Image-to-URL in the deliverable step.
- Source-Test-to-URL in the deliverable step.
- Testing-Image-to-URL in the deliverable step.
- Source-Test-Scan-to-URL in the deliverable step.
- Scanning-Image-Scan-to-URL in the deliverable step.

Creates

A configmap in which the `.data` field has a key `deliverable` for which the value is the YAML definition of a Deliverable.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing the necessary permissions for the Delivery to create children objects. Populates the Deliverable's serviceAccount parameter. The service account must be in the same namespace as the Deliverable.	`- name: serviceAccount` `value: default`
`gitops_ssh_secret`	Name of the secret where credentials exist for fetching the configuration from a Git repository. Populates the Deliverable's gitops_ssh_secret parameter. The service account must be in the same namespace as the Deliverable.	`- name: gitops_ssh_secret` `value: ssh-secret`
`gitops_branch`	Name of the branch from which to fetch the configuration.	`- name: gitops_branch` `value: main`
`gitops_repository`	The full repository URL to which the configuration is fetched. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/cartographer"`
`gitops_repository_prefix`	The prefix of the repository URL. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/"`
`gitops_server_address`	The server URL of the Git repository from which configuration is fetched.	`- name: gitops_server_address` `value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`

Parameter name	Meaning	Example
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`

`registry`	Specification of the registry server and repository from which the configuration is fetched.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`
		`ca_cert_data:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

More information

For information about the ClusterDelivery shipped with `ootb-delivery-basic`, see Out of the Box Delivery Basic.

For information about using the Deliverable object in a multicluster environment, see Getting started with multicluster Tanzu Application Platform.

delivery-source-template

Purpose

Continuously fetches Kubernetes configuration files from a Git repository or container image registry and makes them available on the cluster.

Used by

- Delivery-Basic

Creates

The source-template creates one of three objects, either: - GitRepository. Created if the deliverable has `.spec.source.git` defined. - ImageRepository. Created if the deliverable has `.spec.source.image` defined.

GitRepository

`GitRepository` makes source code from a particular commit available as a tarball in the cluster. Other resources in the supply chain can then access that code.

Parameters

Parameter name	Meaning	Example
`gitImplementation`	The library used to fetch source code. If not provided, Tanzu Application Platform's default implementation uses `go-git`, which works with the providers supported by Tanzu Application Platform: GitHub and GitLab. An alternate value that you can use with other Git providers is `libgit2`.	`- name: gitImplementation value: libgit2`
`gitops_ssh_secret`	Name of the secret used to provide credentials for the Git repository. The secret with this name must exist in the same namespace as the `Deliverable`. The credentials must be sufficient to read the repository. If not provided, Tanzu Application Platform defaults to look for a secret named `git-ssh`. See Git authentication.	`- name: gitops_ssh_secret value: git-credentials`

Note

Some Git providers, notably Azure DevOps, require you to use `libgit2` due to the server-side implementation providing support only for git's v2 protocol. For information about the features supported by each implementation, see git implementation in the flux documentation.

More information

For an example using the Tanzu CLI to create a Workload using GitHub as the provider of source code, see Create a workload from GitHub repository.

For information about GitRepository objects, see GitRepository.

ImageRepository

`ImageRepository` makes the contents of a container image available as a tarball on the cluster.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account, providing credentials to `ImageRepository` for fetching container images. The service account must exist in the same namespace as the Deliverable.	`- name: serviceAccount value: default`

More information

For information about the ImageRepository resource, see ImageRepository reference docs.

app-deploy

Purpose

Applies Kubernetes configuration to the cluster.

Used by

- Delivery-Basic

Creates

A kapp App.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account providing the necessary privileges for `App` to apply the Kubernetes objects to the cluster. The service account must be in the same namespace as the Deliverable.	`- name: serviceAccount` `value: default`
`gitops_sub_path` (deprecated)	Sub directory within the configuration bundle that is used for looking up the files to apply to the Kubernetes cluster. **DEPRECATED**	`- name: gitops_sub_path` `value: ./config`

Note

The `gitops_sub_path` parameter is deprecated. Use `deliverable.spec.source.subPath` instead.

More information

For details about RBAC and how `kapp-controller` makes use of the ServiceAccount provided through the Deliverable's `serviceAccount` parameter, see kapp-controller's Security Model.

carvel-package (experimental)

Purpose

Bundles Kubernetes configuration into a Carvel Package.

Used by

- Source-to-URL-Package (experimental) in the carvel-package step.
- Basic-Image-to-URL-Package (experimental) in the carvel-package step.

Creates

A taskrun.tekton.dev which provides configuration to a Tekton ClusterTask to bundle Kubernetes configuration into a Carvel Package.

This template uses the lifecycle: tekton flag to create new immutable objects rather than updating the previous object.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account to use for providing Docker credentials. The service account must exist in the same namespace as the Workload. The service account must have a secret associated with the credentials. See Configuring authentication for Docker in the Tekton documentation.	`- name: serviceAccount` `value: default`

`registry`	Specification of the registry server and repository in which the built image is placed.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`

`carvel_package_gitops_subpath`	Specifies the subpath to which Carvel Packages should be written.	`- name: carvel_package_gitops_subpath` `value: path/to/my/dir`
`carvel_package_name_suffix`	Specifies the suffix to append to the Carvel Package name. The format is WORKLOAD_NAME.WORKLOAD_NAMESPACE.carvel_package_name_suffix The full Carvel Package name must be a valid DNS subdomain name as defined in RFC 1123.	`- name: carvel_package_name_suffix` `value: vmware.com`

Parameter name	Meaning	Example

`carvel_package_parameters`	Specifies the custom Carvel Package parameters	`- name: carvel_package_parameters`
		`value: |`
		`- selector:`
		`matchLabels:`
		`apps.tanzu.vmware.com/workload-type: server`
		`schema: |`
		`#@data/values-schema`
		`---`
		`#@schema/title "Workload name"`
		`#@schema/desc "Required. Name of the workload, used by K8s Ingress HTTP rules."`
		`#@schema/example "tanzu-java-web-app"`
		`#@schema/validation min_len=1`
		`workload_name: ""`
		
		`#@schema/title "Replicas"`
		`#@schema/desc "Number of replicas."`
		`replicas: 1`
		
		`#@schema/title "Port"`
		`#@schema/desc "Port number for the backend associated with K8s Ingress."`
		`port: 8080`
		
		`#@schema/title "Hostname"`
		`#@schema/desc "If set, K8s Ingress will be created with HTTP rules for hostname."`
		`#@schema/example "app.tanzu.vmware.com"`
		`hostname: ""`
		
		`#@schema/title "Cluster Issuer"`
		`#@schema/desc "CertManager Issuer to use to generate c`

Parameter name	Meaning	Example
		ertificate for K8s Ingress." cluster_issuer: "tap-ingress-selfsigned" overlays: | #@ load("@ytt:overlay", "overlay") #@ load("@ytt:data", "data") #@overlay/match by=overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"}) --- spec: #@overlay/match missing_ok=True replicas: #@ data.values.replicas #@ if data.values.hostname != "": --- apiVersion: networking.k8s.io/v1 kind: Ingress metadata: name: #@ data.values.workload_name annotations: cert-manager.io/cluster-issuer: #@ data.values.cluster_issuer ingress.kubernetes.io/force-ssl-redirect: "true" kubernetes.io/ingress.class: contour kapp.k14s.io/change-rule: "upsert after upserting Services" labels: app.kubernetes.io/component: "run" carto.run/workload-name: #@ data.values.workload_name spec:

Parameter name	Meaning	Example
		tls: - secretName: #@ data.values.workload_name hosts: - #@ data.values.hostname rules: - host: #@ data.values.hostname http: paths: - pathType: Prefix path: / backend: service: name: #@ data.values.workload_name port: number: #@ data.values.port #@ end - selector: matchLabels: apps.tanzu.vmware.com/workload-type: web schema: | #@data/values-schema --- #@schema/validation min_len=1 workload_name: "" overlays: "" - selector: matchLabels: apps.tanzu.vmware.com/workload-type: worker schema: | #@data/values-schema --- #@schema/validation min_len=1 workload_name: "" replicas: 1

Parameter name	Meaning	Example

		`overlays: |`
		`#@ load("@ytt:overlay", "overlay")`
		`#@ load("@ytt:data", "data")`
		`#@overlay/match by=overlay.subset({"apiVersion":"apps/v1", "kind": "Deployment"})`
		`---`
		`spec:`
		`#@overlay/match missing_ok=True`
		`replicas: #@ data.values.replicas`

`carvel_package_openapiv3_enabled`	Specifies whether the Carvel Package should include a generated OpenAPIv3 specification	`- name: carvel_package_openapiv3_enabled` `value: true`

More information

To read more about `lifecycle:tekton`, read Cartographer Lifecycle.

package-config-writer-template (experimental)

Purpose

Persist in an external git repository the Carvel Package Kubernetes configuration passed to the template.

Used by

- Source-to-URL-Package (experimental) in the config-writer step.
- Basic-Image-to-URL-Package (experimental) in the config-writer step.

Creates

A runnable which creates a Tekton TaskRun that refers either to the Tekton Task `git-writer`.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount` `value: default`
`gitops_branch`	Name of the branch to push the configuration to.	`- name: gitops_branch` `value: main`
`gitops_user_name`	User name to use in the commits.	`- name: gitops_user_name` `value: "Alice Lee"`
`gitops_user_email`	User email address to use in the commits.	`- name: gitops_user_email` `value: alice@example.com`
`gitops_commit_message`	Message to write as the body of the commits produced for pushing configuration to the Git repository.	`- name: gitops_commit_message` `value: "ci bump"`
`gitops_repository`	The full repository URL to which the configuration is committed. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/cartographer"`
`gitops_repository_prefix`	The prefix of the repository URL. **DEPRECATED**	`- name: gitops_repository` `value: "https://github.com/vmware-tanzu/"`
`gitops_server_address`	The server URL of the Git repository to which configuration is applied.	`- name: gitops_server_address` `value: "https://github.com/"`

Parameter name	Meaning	Example
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner` `value: vmware-tanzu`
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name` `value: cartographer`

`registry`	Specification of the registry server and repository in which the configuration is placed.	`- name: registry`
		`value:`
		`server: index.docker.io`
		`repository: web-team`
		`ca_cert_data:`
		`-----BEGIN CERTIFICATE-----`
		`MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...`
		`-----END CERTIFICATE-----`

`carvel_package_gitops_subpath`	Specifies the subpath to which Carvel Packages should be written.	`- name: carvel_package_gitops_subpath` `value: path/to/my/dir`
`carvel_package_name_suffix`	Specifies the suffix to append to the Carvel Package name. The format is WORKLOAD_NAME.WORKLOAD_NAMESPACE.carvel_package_name_suffix The full Carvel Package name must be a valid DNS subdomain name as defined in RFC 1123.	`- name: carvel_package_name_suffix` `value: vmware.com`

More information

See Gitops vs RegistryOps for more information about the operation of this template and of the package-config-writer-and-pull-requester-template (experimental).

package-config-writer-and-pull-requester-template (experimental)

Purpose

Persist the passed in Carvel Package Kubernetes configuration to a branch in a repository and open a pull request to another branch. (This process allows for manual review of configuration before deployment to a cluster)

Used by

- Source-to-URL-Package (experimental) in the config-writer step.
- Basic-Image-to-URL-Package (experimental) in the config-writer step.

Creates

A runnable which provides configuration to the ClusterRunTemplate `commit-and-pr-pipelinerun` to create a Tekton TaskRun. The Tekton TaskRun refers to the Tekton Task `commit-and-pr`.

Parameters

Parameter name	Meaning	Example
`serviceAccount`	Name of the service account which provides the credentials to the registry or repository. The service account must exist in the same namespace as the Workload.	`- name: serviceAccount value: default`
`gitops_commit_branch`	Name of the branch to which configuration is pushed.	`- name: gitops_commit_branch value: feature`
`gitops_branch`	Name of the branch to which a pull request is opened.	`- name: gitops_branch value: main`
`gitops_user_name`	User name to use in the commits.	`- name: gitops_user_name value: "Alice Lee"`
`gitops_user_email`	User email address to use in the commits.	`- name: gitops_user_email value: alice@example.com`

Parameter name	Meaning	Example
`gitops_commit_message`	Message to write as the body of the commits produced for pushing configuration to the Git repository.	`- name: gitops_commit_message value: "ci bump"`
`gitops_pull_request_title`	Title of the pull request to be opened.	`- name: gitops_pull_request_title value: "ready for review"`
`gitops_pull_request_body`	Body of the pull request to be opened.	`- name: gitops_pull_request_body value: "generated by supply chain"`
`gitops_server_address`	The server URL of the Git repository to which configuration is applied.	`- name: gitops_server_address value: "https://github.com/"`
`gitops_repository_owner`	The owner/organization to which the repository belongs.	`- name: gitops_repository_owner value: vmware-tanzu`
`gitops_repository_name`	The name of the repository.	`- name: gitops_repository_name value: cartographer`
`gitops_server_kind`	The kind of Git provider	`- name: gitops_server_kind value: gitlab`

Parameter name	Meaning	Example
`carvel_package_gitops_subpath`	Specifies the subpath to which Carvel Packages should be written.	`- name: carvel_package_gitops_subpath value: path/to/my/dir`
`carvel_package_name_suffix`	Specifies the suffix to append to the Carvel Package name. The format is WORKLOAD_NAME.WORKLOAD_NAMESPACE.carvel_package_name_suffix The full Carvel Package name must be a valid DNS subdomain name as defined in RFC 1123.	`- name: carvel_package_name_suffix value: vmware.com`

More information

See Gitops vs RegistryOps for more information about the operation of this template and of the package-config-writer-template (experimental).

ClusterRunTemplate reference for Supply Chain Choreographer

This topic lists the objects you can use with Supply Chain Choreographer. All the objects referenced in this topic are Cartographer ClusterRunTemplates packaged in Out of the Box Templates. This topic describes the one or more objects they create, the supply chains that include them, and the parameters they use.

tekton-source-pipelinerun

Purpose

Tests source code.

Used by

- testing-pipeline

Creates

This ClusterRunTemplate creates a Tekton PipelineRun referring to the user's Tekton Pipeline.

Inputs

ClusterRunTemplate reference for Supply Chain Choreographer

Input name	Meaning	Example

`tekton-params`	Set of parameters to pass to the Tekton Pipeline	`- name: source-url`
		`value: https://github.com/vmware-tanzu/cartographer.git`
		`- name: source-revision`
		`value: e4a53f49a92fc913d26f8cc23d59102a51a5e635`
		`- name: verbose`
		`value: true`
		`- name: foo`
		`value: bar`

More information

For information about the runnable created in the OOTB Testing and OOTB Testing and Scanning, see testing-pipeline.

For information about the Tekton Pipeline that the user must create, see Tekton/Pipeline.

tekton-taskrun

Purpose

Generic template for creating a Tekton TaskRun.

Used by

- config-writer-template

Creates

A Tekton TaskRun.

Inputs

Input name	Meaning	Example
`serviceAccount`	Service Account with permissions necessary for the Tekton Task	default
`taskRef`	Reference to the Tekton Task to which the TaskRun provides parameters	`kind: ClusterTask` `name: git-writer`

`params`	Parameters which are provided to the Tekton Task	`- name: git_branch`
		`value: main`
		`- name: git_user_name`
		`value: "Some Name"`

commit-and-pr-pipelinerun

Purpose

Commit configuration to a Git repository and open a pull request for review.

Used by

- config-writer-and-pull-requester-template

Creates

Creates a Tekton TaskRun referring to the `commit-and-pr` Tekton Task.

Inputs

Input name	Meaning	Example
`serviceAccount`	Service Account with credentials for the Git repository	`default`
`git_server_kind`	Type of Git provider	`github`
`git_server_address`	Server URL	`https://github.com`
`repository_owner`	Owner or Organization in which the repository resides	`vmware-tanzu`
`repository_name`	Name of the repository	`cartographer`
`commit_branch`	Name of the commit branch. Recommended value is an empty string.	`""`
`pull_request_title`	Title of the PR to be opened	`"Update"`
`pull_request_body`	Body of the PR to be opened	`"Ready for review"`
`base_branch`	Branch into which the PR is merged	`main`
`git_user_name`	User name associated with the commit	`Waciuma Rasheed`
`git_user_email`	User email associated with the commit	`Sam@todd.com`
`git_commit_message`	Message on commit	`"App update"`
`git_files`	Base64 encoded JSON file where keys equal the filename and the value is the file contents.	`"eyJkZWxpdmVyeS55bWwiOiJhcGlWZXJzaW9uOiBzZXJ2aW5nLmtuYXRpdmUuZGV2L3YxXG5raW5kOiBTZXJ2aWNlXG4ifQ=="`

`sub_path`	The directory location in the repository in which to write the files.	`"."`

More information

For information about the template creating the related runnable, see config-writer-and-pull-requester-template.

For information about gitops, see GitOps versus RegistryOps.

Delivery reference for Supply Chain Choreographer

This topic describes the delivery parameters and templates you can use with Supply Chain Choreographer.

Tanzu Application Platform delivery package installs a single ClusterDelivery.

The delivery provides some parameters to the templates. The parameters provided by the deliverable might override some of the delivery parameters in this topic. For more information about how parameters work, including precedence rules, see the Cartographer documentation.

delivery-basic

Purpose

- Fetches Kubernetes configuration created by a supply chain.
- Deploys the configuration on the cluster.

Resources

The following resources describe the templates.

source-provider

Refers to delivery-source-template.

Parameters provided:

- `serviceAccount` from tap-value `service_account`. Overridable by deliverable.
- `gitImplementation` from tap-value `git_implementation`. Not overridable by deliverable.

Deployer

Refers to app-deploy template.

Parameter provided:

- `serviceAccount` from tap-value `service_account`. Overridable by deliverable.

Package

Refers to Out of the Box Delivery Basic.

More information

For information about setting `tap-values.yaml` at installation time, see Install Out of the Box Delivery Basic.

Use Git with Supply Chain Choreographer

This topic explains how you can use Git with Supply Chain Choreographer.

The out of the box supply chains and delivery use Git in three ways:

- To fetch the developers source code, using the template.
- To store complete Kubernetes configuration, the write side of GitOps, using template 1, template 2, template 3 (experimental), and template 4 (experimental).
- To fetch stored Kubernetes configuration, the read side of GitOps, from either the same or a different Kubernetes cluster, using the template.

Supported Git Repositories

Tanzu Application Platform supports three git providers:

- Github
- Gitlab
- Azure DevOps

Related Articles

Git Authentication walks through the objects, such as secrets and service accounts, to create on cluster to allow supply chain Git operations to succeed.

GitOps versus RegistryOps discusses the two methods of storing built Kubernetes configuration, either in a Git repository or a container image registry, and walks through the parameters that must be provided for each.

Configuration for Azure DevOps: discusses configuration necessary for working with this git provider.

Use GitOps or RegistryOps with Supply Chain Choreographer

You can use GitOps or RegistryOps to manage your Kubernetes configuration with Supply Chain Choreographer.

Regardless of the supply chain that a workload goes through, in the end, some Kubernetes configuration is pushed to an external entity, either to a Git repository or to a container image registry.

For example:

```
Supply Chain

  -- fetch source
    -- test
      -- build
        -- scan
          -- apply-conventions
            -- push config        * either to Git or Registry
```

This topic dives into the specifics of that last phase of the supply chains by pushing configuration to a Git repository or a container image registry.

Note

For more information about providing source code either from a local directory or Git repository, see Building from Source.

GitOps

The GitOps approach differs from local iteration in that GitOps configures the supply chains to push the Kubernetes configuration to a remote Git repository. This allows users to compare configuration changes and promote those changes through environments by using GitOps principles.

Typically associated with an outerloop workflow, the GitOps approach is only activated if a collection of parameters are set:

- `gitops.server_address` during the Out of the Box Supply Chains package installation or `gitops_server_address` configured as a workload parameter.
- `gitops.repository_owner` during the Out of the Box Supply Chains package installation or `gitops_repository_owner` configured as a workload parameter.
- `gitops.repository_name` during the Out of the Box Supply Chains package installation or `gitops_repository_name` configured as a workload parameter.

With all three values set, Kubernetes configuration is written to the specified repository. If a value is set at installation and the corresponding workload parameter is also set, the value of the workload parameter is respected.

In the repository, files are located in the `./config/{workload-namespace}/{workload-name}` directory. This allows multiple workloads to commit configuration to the same repository.

Examples

`tap-values.yaml`

```
gitops:
  server_address:
  repository_owner:
  repository_name:
```

`workload`

```
  name: incrediApp
  namespace: awesomeTeam
  params:
    - name: gitops_server_address
      value: https://github.com/
    - name: gitops_repository_owner
      value: vmware-tanzu
    - name: gitops_repository_name
      value: cartographer
```

Resulting gitops repository: `https://github.com/vmware-tanzu/cartographer`

Directory containing configuration: `./config/awesomeTeam/incrediApp`

`tap-values.yaml`

```
gitops:
  server_address: https://github.com/
```

```
  repository_owner: vmware-tanzu
  repository_name: cartographer
```

`workload`

```
  name: superApp
  namespace: awesomeTeam
```

Resulting gitops repository: `https://github.com/vmware-tanzu/cartographer`

Directory containing configuration: `./config/awesomeTeam/superApp`

`tap-values.yaml`

```
gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
```

`workload`

```
  name: superApp
  namespace: awesomeTeam
  params:
     - name: gitops_repository_owner
       value: buildpacks-community
     - name: gitops_repository_name
       value: kpack
```

Resulting GitOps repository: `https://github.com/buildpacks-community/kpack`

Directory containing configuration: `./config/awesomeTeam/superApp`

`tap-values.yaml`

```
gitops:
  server_address:
  repository_owner:
  repository_name:
```

`workload`

```
  name: superApp
  namespace: awesomeTeam
  params:
     - name: gitops_repository_owner
       value: buildpacks-community
     - name: gitops_repository_name
       value: kpack
```

Resulting gitops repository: Fails to resolve as some, but not all, of the three required values are provided.

Deprecated parameters

The following parameters are deprecated and no longer recommended for specifying gitops repositories:

- `gitops.repository_prefix`: configured during the Out of the Box Supply Chains package installation.
- `gitops_repository`: configured as a workload parameter.

For example, assuming the installation of the supply chain packages through Tanzu Application Platform profiles and a `tap-values.yaml`:

```
ootb_supply_chain_basic:
  registry:
    server: REGISTRY-SERVER
    repository: REGISTRY-REPOSITORY

  gitops:
    repository_prefix: https://github.com/my-org/
```

Workloads in the cluster with the Kubernetes configuration produced throughout the supply chain are pushed to the repository whose name is formed by concatenating `gitops.repository_prefix` with the name of the workload. In this case, for example, `https://github.com/my-org/$(workload.metadata.name).git`.

```
Supply Chain
  params:
      - gitops_repository_prefix: GIT-REPO_PREFIX


workload-1:
   `git push` to GIT-REPO-PREFIX/workload-1.git

workload-2:
   `git push` to GIT-REPO-PREFIX/workload-2.git

...

workload-n:
   `git push` to GIT-REPO-PREFIX/workload-n.git
```

Alternatively, you can force a workload to publish the configuration in a Git repository by providing the `gitops_repository` parameter to the workload:

```
tanzu apps workload create tanzu-java-web-app \
  --app tanzu-java-web-app \
  --type web \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --git-branch main \
  --param gitops_ssh_secret=GIT-SECRET-NAME \
  --param gitops_repository=https://github.com/my-org/config-repo
```

In this case, at the end of the supply chain, the configuration for this workload is published to the repository provided under the `gitops_repository` parameter.

If you use deprecated parameters, Kubernetes configuration is committed to the `./config` directory in the repository. This can lead to collisions if two workloads specify the same repository, or two workloads in different namespaces have the same name and the `gitops.repository_prefix` is set in `tap-values.yaml`.

If the deprecated values are set and any of the suggested gitops values are set, the deprecated values are ignored.

Examples

`tap-values.yaml`

```
gitops:
```

```
  repository_prefix: https://github.com/vmware-tanzu
```

`workload`

```
  name: superApp
  namespace: awesomeTeam
```

Resulting gitops repository: `https://github.com/vmware-tanzu/incrediApp`

Directory containing configuration: `./config`

`tap-values.yaml`

```
gitops:
  server_address: https://github.com/
  repository_owner: vmware-tanzu
  repository_name: cartographer
```

`workload`

```
  name: superApp
  namespace: awesomeTeam
  params:
    - name: gitops_repository
      value: https://github.com/buildpacks-community/kpack
```

Resulting gitops repository: `https://github.com/vmware-tanzu/cartographer` (The deprecated param `gitops_repository` is ignored.)

Directory containing configuration: `./config/awesomeTeam/superApp`

`tap-values.yaml`

```
gitops:
  repository_prefix: https://github.com/vmware-tanzu
```

`workload`

```
  name: superApp
  namespace: awesomeTeam
  params:
    - name: gitops_repository_owner
      value: buildpacks-community
    - name: gitops_repository_name
      value: kpack
```

Resulting gitops repository: Fails to resolve as some, but not all, of the three gitops values are provided. (The deprecated value repository_prefix is ignored because suggested values are present)

Pull requests

In the standard `git-ops` approach, configuration is pushed to a repository and is immediately applied to a cluster by any deliverable watching that repository. Operators might want to manually review configuration before applying it to the cluster. To do this, operators must specify a `pull_request` commit strategy. You can use this strategy with the following Git providers:

- GitHub
- GitLab

- Azure DevOps

Authentication

The pull request approach requires HTTP(S) authentication with a token.

The pull request function is not a part of the Git specification, but most Git server providers include it. You must authenticate with those providers using a token.

In the Kubernetes secret that holds the Git credentials, the password text box must contain a token. When generating a token, ensure that it has the proper scope:

- On GitHub, the token must have a Repo scope.
- On GitLab, the token must have an API scope.

To use the `pull_request` commit strategy, set the following parameters:

- `commit_strategy` == `pull_request` configured during the Out of the Box Supply Chains package installation.
- `gitops.pull_request.server_kind` configured during the Out of the Box Supply Chains package installation or `gitops_server_kind` configured as a workload parameter. Supported values are `github`, `gitlab`, and `azure`.
- `gitops.pull_request.commit_branch` configured during the Out of the Box Supply Chains package installation or `gitops_commit_branch` configured as a workload parameter.
- `gitops.pull_request.pull_request_title` configured during the Out of the Box Supply Chains package installation or `gitops_pull_request_title` configured as a workload parameter.
- `gitops.pull_request.pull_request_body` configured during the Out of the Box Supply Chains package installation or `gitops_pull_request_body` configured as a workload parameter.

If a value is set at both installation and in a workload parameter, the workload parameter is respected.

The recommended value for `commit_branch` is an empty string. This generates a new branch for each commit based on a hash of the time when the commit is created. This prevents collisions between multiple workloads using a single Git repository.

For example, using the following Tanzu Application Platform values:

```
ootb_supply_chain_basic:
   gitops:
     server_address: https://github.com/
     repository_owner: vmware-tanzu
     repository_name: cartographer
     branch: main
     commit_strategy: pull_request
     pull_request:
       server_kind: github
       commit_branch: ""
       pull_request_title: ready for review
       pull_request_body: generated by supply chain
```

In a workload with the name `app` in the `dev` namespace, you find:

A commit to the `https://github.com/vmware-tanzu/cartographer` repository on a branch with a random name. For example, `MTY1MTYxMzE0NQo=`. There is a pull request open to merge this branch into the base branch `main`.

Authentication

Regardless of how the supply chains are configured, if the repository prefix or repository name is configured to push to Git, you must provide credentials for the remote provider by using a Kubernetes secret in the same namespace as the workload attached to the workload `ServiceAccount`.

Because the operation of pushing requires elevated permissions, credentials are required by both public and private repositories.

HTTP(S) Basic-auth or Token-based authentication

If the repository at which configuration is published uses `https://` or `http://` as the URL scheme, the Kubernetes secret must provide the credentials for that repository as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: GIT-SECRET-NAME  # `git-ssh` is the default name.
                         #   - operators can change such default by using the
                         #     `gitops.ssh_secret` property in `tap-values.yaml`
                         #   - developers can override by using the workload parameter
                         #     named `gitops_ssh_secret`.
  annotations:
    tekton.dev/git-0: GIT-SERVER        # ! required
type: kubernetes.io/basic-auth          # ! required
stringData:
  username: GIT-USERNAME
  password: GIT-PASSWORD
```

Both the Tekton annotation and the `basic-auth` secret type must be set. `GIT-SERVER` must be prefixed with the appropriate URL scheme and the Git server. For example, for `https://github.com/vmware-tanzu/cartographer`, `https://github.com` must be provided as the `GIT-SERVER`.

To use the pull request approach, the password text box must contain a token. See Pull Requests.

After the `Secret` is created, attach it to the `ServiceAccount` used by the workload. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

For more information about the credentials and setting up the Kubernetes secret, see Git Authentication's HTTP section.

SSH

If the repository to which configuration is published uses `https://` or `http://` as the URL scheme, the Kubernetes secret must provide the credentials for that repository as follows:

```
apiVersion: v1
kind: Secret
metadata:
```

```
  name: GIT-SECRET-NAME  # `git-ssh` is the default name.
                         #   - operators can change such default through the
                         #     `gitops.ssh_secret` property in `tap-values.yaml`
                         #   - developers can override by using the workload parameter
                         #     named `gitops_ssh_secret`.
  annotations:
    tekton.dev/git-0: GIT-SERVER
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: SSH-PRIVATE-KEY     # private key with push-permissions
  identity: SSH-PRIVATE-KEY           # private key with pull permissions
  identity.pub: SSH-PUBLIC-KEY        # public of the `identity` private key
  known_hosts: GIT-SERVER-PUBLIC-KEYS # git server public keys
```

After the `Secret` is created, attach it to the `ServiceAccount` used by the workload. For example:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
secrets:
  - name: registry-credentials
  - name: tap-registry
  - name: GIT-SECRET-NAME
imagePullSecrets:
  - name: registry-credentials
  - name: tap-registry
```

For information about the credentials and setting up the Kubernetes secret, see Git Authentication's SSH section.

GitOps workload parameters

While installing `ootb-*`, operators can configure `gitops.repository_prefix` to indicate what prefix the supply chain must use when forming the name of the repository to push to the Kubernetes configurations produced by the supply chains.

To change the behavior to use GitOps, set the source of the source code to a Git repository. As the supply chain progresses, configuration is pushed to a repository named `$(gitops.repository_prefix) + $(workload.name)`.

For example, configure `gitops.repository_prefix` to `git@github.com/foo/` and create a workload as follows:

```
tanzu apps workload create tanzu-java-web-app \
  --git-branch main \
  --git-repo https://github.com/vmware-tanzu/application-accelerator-samples \
  --sub-path tanzu-java-web-app \
  --label app.kubernetes.io/part-of=tanzu-java-web-app \
  --type web
```

Expect to see the following output:

```
Create workload:
      1 + |---
      2 + |apiVersion: carto.run/v1alpha1
      3 + |kind: Workload
      4 + |metadata:
      5 + |  labels:
      6 + |    apps.tanzu.vmware.com/workload-type: web
      7 + |    app.kubernetes.io/part-of: tanzu-java-web-app
      8 + |  name: tanzu-java-web-app
      9 + |  namespace: default
```

```
    10 + |spec:
    11 + |  source:
    12 + |    git:
    13 + |      ref:
    14 + |        branch: main
    15 + |      url: https://github.com/vmware-tanzu/application-accelerator-samples
    16 + |    subPath: tanzu-java-web-app
```

As a result, the Kubernetes configuration is pushed to `git@github.com/foo/tanzu-java-web-app.git`.

Regardless of the setup, developers can also manually override the repository where configuration is pushed to by tweaking the following parameters:

- `gitops_ssh_secret`: Name of the secret in the same namespace as the workload where SSH credentials exist for pushing the configuration produced by the supply chain to a Git repository. Example: `ssh-secret`
- `gitops_repository`: SSH URL of the Git repository to push the Kubernetes configuration produced by the supply chain to. Example: `ssh://git@foo.com/staging.git`
- `gitops_branch`: Name of the branch to push the configuration to. Example: `main`
- `gitops_commit_message`: Message to write as the body of the commits produced for pushing configuration to the Git repository. Example: `ci bump`
- `gitops_user_name`: User name to use in the commits. Example: `Alice Lee`
- `gitops_user_email`: User email address to use in the commits. Example: `alice@example.com`

Read more on Git

See Git Reference

RegistryOps

RegistryOps is typically used for inner loop flows where configuration is treated as an artifact from quick iterations by developers. In this scenario, at the end of the supply chain, configuration is pushed to a container image registry in the form of an imgpkg bundle. You can think of it as a container image whose sole purpose is to carry arbitrary files.

To enable this mode of operation, the supply chains must be configured **without** the following parameters being configured during the installation of the `ootb-` packages or overwritten by the workload by using the following parameters:

- `gitops_repository_prefix`
- `gitops_repository`

If none of the parameters are set, the configuration is pushed to the same container image registry as the application image. That is, to the registry configured under the `registry: {}` section of the `ootb-` values.

For example, assuming the installation of Tanzu Application Platform by using profiles, configure the `ootb-supply-chain*` package as follows:

```
ootb_supply_chain_basic:
  registry:
    server: REGISTRY-SERVER
    repository: REGISTRY_REPOSITORY
```

The Kubernetes configuration produced by the supply chain is pushed to an image named after `REGISTRY-SERVER/REGISTRY-REPOSITORY` including the workload name.

In this scenario, no extra credentials must be set up, because the secret containing the credentials for the container image registry were already configured during the setup of the workload namespace.

Overview of Supply Chain Security Tools for VMware Tanzu - Policy Controller

Supply Chain Security Tools - Policy Controller is a security tool that helps you ensure that the container images in their registry have not been tampered with. Policy Controller is a Kubernetes Admission Controller that allows you to apply policies to verify signatures on container images before being admitted to a cluster.

The Policy Controller:

- Verifies signatures on container images used by Kubernetes resources
- Enforces policies to allow or deny images being admitted a cluster
- Allows operators to define multiple policies in the cluster
- Allows operators to select which `namespaces` to enforce policies against
- Supports `cosign` signatures and keyless signing
- Supports storing public keys in a KMS

It enforces its policies against all resources that create `Pod`s as part of their life cycle:

- `Pod`
- `ReplicaSet`
- `Deployment`
- `Job`
- `StatefulSet`
- `DaemonSet`
- `CronJob`

Note

This component is the successor to `Supply Chain Security Tools - Sign`, which is deprecated. Support and maintenance for `Supply Chain Security Tools - Sign` continues. Monitor Release Notes for updates.

Supply Chain Security Tools - Policy Controller is based on Sigstore's Policy Controller and is compatible only with `cosign` signatures. See Cosign and Policy Controller in GitHub. For information about image signing and verification, see Sigstore open source community and the cosign project in GitHub.

The Policy Controller component is a policy enforcement tool only. It does not sign images. Operators can configure image signing for their containers in several ways, including:

- By using Tanzu Build Service
- By using kpack
- By integrating cosign into their build pipelines

Image signatures generated by `cosign` are stored in the same registry location as the image itself unless configured with the `COSIGN_REPOSITORY` environment variable. Policy Controller uses registry

credentials provided in the admission request, Service Account, or `signaturePullSecrets` defined in the policy to connect to the registry to verify a signature.

Important

This component does not work with insecure registries.

To Install Supply Chain Security Tools - Policy Controller, see Install Supply Chain Security Tools - Policy Controller

Overview of Supply Chain Security Tools for VMware Tanzu - Policy Controller

Supply Chain Security Tools - Policy Controller is a security tool that helps you ensure that the container images in their registry have not been tampered with. Policy Controller is a Kubernetes Admission Controller that allows you to apply policies to verify signatures on container images before being admitted to a cluster.

The Policy Controller:

- Verifies signatures on container images used by Kubernetes resources
- Enforces policies to allow or deny images being admitted a cluster
- Allows operators to define multiple policies in the cluster
- Allows operators to select which `namespaces` to enforce policies against
- Supports `cosign` signatures and keyless signing
- Supports storing public keys in a KMS

It enforces its policies against all resources that create `Pod`s as part of their life cycle:

- `Pod`
- `ReplicaSet`
- `Deployment`
- `Job`
- `StatefulSet`
- `DaemonSet`
- `CronJob`

Note

This component is the successor to `Supply Chain Security Tools - Sign`, which is deprecated. Support and maintenance for `Supply Chain Security Tools - Sign` continues. Monitor Release Notes for updates.

Supply Chain Security Tools - Policy Controller is based on Sigstore's Policy Controller and is compatible only with `cosign` signatures. See Cosign and Policy Controller in GitHub. For information about image signing and verification, see Sigstore open source community and the cosign project in GitHub.

The Policy Controller component is a policy enforcement tool only. It does not sign images. Operators can configure image signing for their containers in several ways, including:

- By using Tanzu Build Service
- By using kpack
- By integrating cosign into their build pipelines

Image signatures generated by `cosign` are stored in the same registry location as the image itself unless configured with the `COSIGN_REPOSITORY` environment variable. Policy Controller uses registry credentials provided in the admission request, Service Account, or `signaturePullSecrets` defined in the policy to connect to the registry to verify a signature.

Important

This component does not work with insecure registries.

To Install Supply Chain Security Tools - Policy Controller, see Install Supply Chain Security Tools - Policy Controller

Install Supply Chain Security Tools - Policy Controller

You install Supply Chain Security Tools - Policy Controller as part of Tanzu Application Platform's Full, Iterate, and Run profiles. You can use the instructions in this topic to manually install SCST - Policy Controller.

Note

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Security Tools - Policy Controller. For more information about profiles, see Components and installation profiles.

Prerequisites

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- A container image registry that supports TLS connections.

Important

This component does not work with not secure registries.

- For keyless authorities support, you must set `policy.tuf_enabled: true`. By default, the public official Sigstore The Update Framework (TUF) server is used. To target an alternative Sigstore stack, specify `policy.tuf_mirror` and `policy.tuf_root`.
- If you are installing in an air-gapped environment and require keyless authorities, you must deploy a Sigstore Stack on the cluster or be accessible from the air-gapped environment.
- During configuration, you provide a cosign public key to validate signed images. The Policy Controller only supports ECDSA public keys. An example cosign public key is provided that can validate an image from the public cosign registry. To provide your own key and images, follow the Cosign Quick Start Guide in GitHub.

Caution

This component rejects `pods` if they are not correctly configured. Test your configuration in a test environment before applying policies to your production cluster.

Install

To install Supply Chain Security Tools - Policy Controller:

1. List version information for the package by running:

```
tanzu package available list policy.apps.tanzu.vmware.com --namespace tap-insta
ll
```

For example:

```
$ tanzu package available list policy.apps.tanzu.vmware.com --namespace tap-ins
tall
- Retrieving package versions for policy.apps.tanzu.vmware.com...
  NAME                           VERSION        RELEASED-AT
  policy.apps.tanzu.vmware.com  1.2.0          2023-10-01 20:00:00 -0400 EDT
```

2. (Optional) Make changes to the default installation settings by running:

```
tanzu package available get policy.apps.tanzu.vmware.com/VERSION --values-schem
a --namespace tap-install
```

Where `VERSION` is the version number you discovered. For example, `1.2.0`.

For example:

```
$ tanzu package available get policy.apps.tanzu.vmware.com/1.2.0 --values-schem
a --namespace tap-install
| Retrieving package details for policy.apps.tanzu.vmware.com/1.2.0...

KEY                         DEFAULT        TYPE     DESCRIPTION
custom_cas                  <nil>          array    List of custom CA contents t
hat should be included in the application container for registry communication.
                                                    An array of items containing
a ca_content field with the PEM-encoded contents of a certificate authority.
deployment_namespace        cosign-system  string   Deployment namespace specifi
es the namespace where this component should be deployed to.
                                                    If not specified, "cosign-sy
stem" is assumed.
fail_on_empty_authorities  true           boolean  Configure if a ClusterImageP
olicy will fail or allow empty authorities
limits_cpu                  200m           string   The CPU limit defines a hard
ceiling on how much CPU time
                                                    that the Policy Controller m
anager container can use.
                                                    https://kubernetes.io/docs/c
oncepts/configuration/manage-resources-containers/#meaning-of-cpu

no_match_policy             deny           string   The action when no policy ma
tches the admitting image digest. Valid values are "warn", "allow", or "deny".
quota.pod_number            6              string   The maximum number of Policy
Controller Pods allowed to be created with the priority class
                                                    system-cluster-critical. Thi
s value must be enclosed in quotes (""). If this value is not
                                                    specified then a default val
ue of 6 is used.
replicas                    1              integer  The number of replicas to be
created for the Policy Controller. This value must not be enclosed
```

```
                                                  in quotes. If this value is
not specified then a default value of 1 is used.
requests_memory            20Mi           string   The memory request defines t
he minium memory amount for the Policy Controller manager.
                                                  https://kubernetes.io/docs/c
oncepts/configuration/manage-resources-containers/#meaning-of-memory

tuf_root                                  string   The root.json file content o
f the TUF mirror

custom_ca_secrets          <nil>          array    List of custom CA secrets th
at should be included in the application container for registry communication.
                                                  An array of secret reference
s each containing a secret_name field with the secret name to be referenced
                                                  and a namespace field with t
he name of the namespace where the referred secret resides.
limits_memory              200Mi          string   The memory limit defines a h
ard ceiling on how much memory
                                                  that the Policy Controller m
anager container can use.
                                                  https://kubernetes.io/docs/c
oncepts/configuration/manage-resources-containers/#meaning-of-memory

requests_cpu               20m            string   The CPU request defines the
minimum CPU time for the Policy
                                                  Controller manager. During C
PU contention, CPU request is used as
                                                  a weighting where higher CPU
requests are allocated more CPU time.
                                                  https://kubernetes.io/docs/c
oncepts/configuration/manage-resources-containers/#meaning-of-cpu

tuf_mirror                                string   TUF mirror address
```

3. Create a file named `scst-policy-values.yaml` and add the settings you want to customize:

    - `custom_ca_secrets`: If your container registries are secured by self-signed certificates, this setting controls which secrets are added to the application container as custom certificate authorities (CAs). `custom_ca_secrets` consists of an array of items. Each item contains two text boxes: the `secret_name` text box defines the name of the secret, and the `namespace` text box defines the name of the namespace where said secret is stored.

      For example:

      ```
      custom_ca_secrets:
      - secret_name: first-ca
        namespace: ca-namespace
      - secret_name: second-ca
        namespace: ca-namespace
      ```

      Note

      This setting is allowed even if `custom_cas` is defined.

    - `custom_cas`: This setting enables adding certificate content in PEM format. The certificate content is added to the application container as custom certificate authorities (CAs) to communicate with registries deployed with self-signed certificates. `custom_cas` consists of an array of items. Each item contains a single text box named `ca_content`. The value of this text box must be a PEM-formatted certificate authority. The certificate content must be defined as a YAML block,

preceded by the literal indicator (`|`) to preserve line breaks and ensure that the certificates are interpreted correctly.

For example:

```
custom_cas:
- ca_content: |
    ----- BEGIN CERTIFICATE -----
    first certificate content here...
    ----- END CERTIFICATE -----
- ca_content: |
    ----- BEGIN CERTIFICATE -----
    second certificate content here...
    ----- END CERTIFICATE -----
```

Note

This setting is allowed even if `custom_ca_secrets` is defined.

- `deployment_namespace`: This setting controls the namespace to which this component is deployed. When not specified, the namespace `cosign-system` is assumed. This component creates the specified namespace to deploy required resources. Select a namespace that is not used by any other components.
- `limits_cpu`: This setting controls the maximum CPU resource allocated to the Policy admission controller. The default value is “200m”. See Kubernetes documentation.
- `limits_memory`: This setting controls the maximum memory resource allocated to the Policy admission controller. The default value is “200Mi”. See Kubernetes documentation.
- `quota.pod_number`: This setting controls the maximum number of pods that are allowed in the deployment namespace with the `system-cluster-critical` priority class. This priority class is added to the pods to prevent preemption of this component’s pods in case of node pressure.

  The default value for this text box is `6`. If your use requires more than 6 pods, change this value to allow the number of replicas you intend to deploy.

  Note

  VMware recommends to run this component with a critical priority level to prevent the cluster from rejecting all admission requests if the component’s `pod`s are evicted due to resource limits.

- `replicas`: This setting controls the default amount of replicas deployed by this component. The default value is `1`.

  For production environments: VMware recommends you increase the number of replicas to `3` to ensure that the availability of the component and better admission performance.

- `requests_cpu`: This setting controls the minimum CPU resource allocated to the Policy admission controller. During CPU contention, this value is used as a weighting where higher values indicate more CPU time is allocated. The default value is `20m`. See CPU resource units in the Kubernetes documentation.

- `requests_memory`: This setting controls the minimum memory resource allocated to the Policy admission controller. The default value is `20Mi`. See Memory resource units in the Kubernetes documentation.
- `tuf_enabled`: This setting defines whether the TUF initialization is done on startup. It is required for keyless verification support. The default value is `false`, which means that keyless authorities of `ClusterImagePolicy` are not supported. Also, policy-controller does not have an external dependency on setup.
- `tuf_root`: The root.json file content of the TUF mirror.
- `tuf_mirror`: This setting defines the TUF mirror address which is used for doing the initialization.
- `no_match_policy`: The action when no policy matches the admitting image digest. Valid values are `"warn"`, `"allow"`, or `"deny"`. Default value is `"deny"`
- `fail_on_empty_authorities`: Failing or allowing empty authorities when adding a new `ClusterImagePolicy`. Default value is `true`.

4. Install the package:

```
tanzu package install policy-controller \
  --package policy.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file scst-policy-values.yaml
```

Where `VERSION` is the version number you discovered earlier. For example, `1.2.0`.

For example:

```
$ tanzu package install policy-controller \
    --package policy.apps.tanzu.vmware.com \
    --version 1.2.0 \
    --namespace tap-install \
    --values-file scst-policy-values.yaml

  Installing package 'policy.apps.tanzu.vmware.com'
  Getting package metadata for 'policy.apps.tanzu.vmware.com'
  Creating service account 'policy-controller-tap-install-sa'
  Creating cluster admin role 'policy-controller-tap-install-cluster-role'
  Creating cluster role binding 'policy-controller-tap-install-cluster-rolebind
ing'
  Creating package resource
  Waiting for 'PackageInstall' reconciliation for 'policy-controller'
  'PackageInstall' resource install status: Reconciling
  'PackageInstall' resource install status: ReconcileSucceeded
  'PackageInstall' resource successfully reconciled

  Added installed package 'policy-controller'
```

After you run the commands earlier the policy controller is running.

Policy Controller is now installed, but it does not enforce any policies by default. Policies must be explicitly configured on the cluster. To configure signature verification policies, see Configuring Supply Chain Security Tools - Policy.

Migration From Supply Chain Security Tools - Sign

This topic explains how you can migrate from Supply Chain Security Tools - Sign to Supply Chain Security Tools - Policy.

In Tanzu Application Platform v1.4, the Image Policy Webhook is removed. If the Image Policy Webhook was used with the previous Tanzu Application Platform versions in your cluster, you must migrate the `ClusterImagePolicy` resource from Image Policy Webhook to Policy Controller. For information about additional features introduced in SCST - Policy, see Configuring Supply Chain Security Tools - Policy.

Enable Policy Controller on Namespaces

Policy Controller works with an opt-in system. Operators must update namespaces with the label `policy.sigstore.dev/include: "true"` to the namespace resource to enable Policy Controller verification.

```
kubectl label namespace my-secure-namespace policy.sigstore.dev/include=true
```


Caution

Without a Policy Controller ClusterImagePolicy applied, there are fallback behaviors where images are validated against the public Sigstore Rekor and Fulcio servers by using a keyless authority flow. Therefore, if the deploying image is signed publicly by a third-party using the keyless authority flow, the image are admitted as it can validate against the public Rekor and Fulcio. To avoid this behavior, develop, and apply a ClusterImagePolicy that applies to the images being deployed in the namespace.

Policy Controller ClusterImagePolicy

The Policy Controller `ClusterImagePolicy` does not have a name. Image Policy Controller required that the `ClusterImagePolicy` be named `image-policy` and that there be only one `ClusterImagePolicy`. Multiple Policy Controller `ClusterImagePolicies` are applied. During validation, all `ClusterImagePolicy` that have an image `glob` pattern that matches the deploying image is evaluated. All matched `ClusterImagePolicies` must be valid. For a `ClusterImagePolicy` to be valid, at least one authority in the policy must validate the signature of the deploying image.

Excluding Namespaces

The namespaces listed in `spec.verification.exclude.resources.namespaces[]` must have `policy.sigstore.dev/include` set to `false` or not be set. Therefore, they are exempted from Policy Controller validation.

Image Policy Webhook:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    ...

    exclude:
      resources:
        namespaces:
        - image-policy-system
        - kube-system
```

```
        - cert-manager

    ...
```

Specifying Public Keys

`spec.verification.keys[].publicKey` from Image Policy Webhook is mapped to `spec.authorities[].key.data` for Policy Controller.

The `name` associated with each `key` is no longer required. Image Policy Webhook has direct association between `key` name and `imagePattern`. For Policy Controller, multiple `ClusterImagePolicy` resources are defined to create direct association between image patterns and key authorities.

Image patterns and keys are scoped to each `ClusterImagePolicy` resource.

Therefore, to have direct association be isolated between `key` and `imagePattern`, multiple Policy Controller `ClusterImagePolicy` must be created. Each `ClusterImagePolicy` has the image glob pattern defined and the associated key authorities defined.

Image Policy Webhook:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    ...

    keys:
    - name: official-cosign-key
      publicKey: |
        -----BEGIN PUBLIC KEY-----
        MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRklnt
        IqozONbbdbqz11hlRJy9c7SG+hdcFl9jE9uE/dwtuwU2MqU9T/cN0YkWww==
        -----END PUBLIC KEY-----

    ...
```

Policy Controller:

```
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  authorities:
  ...

  - key:
      data: |
        -----BEGIN PUBLIC KEY-----
        MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRklnt
        IqozONbbdbqz11hlRJy9c7SG+hdcFl9jE9uE/dwtuwU2MqU9T/cN0YkWww==
        -----END PUBLIC KEY-----

  ...
```

Where `POLICY-NAME` is the name of the cluster image policy you want to use.

Specifying Image Matching

`spec.verification.images[].namePattern` from Image Policy Webhook maps to `spec.images[].glob` for Policy Controller.

Policy Controller follows more closely to `glob` matching. For the Image Policy Webhook, `registry.com/*` wildcards all projects and images under the registry. However, `glob` matching uses `/` separator delimiting. Therefore, the `glob` wildcard matching equivalent is `registry.com/**/*`. The `**` allows for recursive project path matching while the trailing `*` images found in the terminating project path.

If only one level of pathing is required, the `glob` pattern is `registry.com/*/*`.

Policy Controller has defaults defined. If `*` is specified, the `glob` matching behavior is `index.docker.io/library/*`. If `*/*` is specified, the `glob` matching behavior is `index.docker.io/*/*`. With these defaults, the `glob` pattern `**` matches against all images.

Image Policy Webhook:

```
---
apiVersion: signing.apps.tanzu.vmware.com/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  verification:
    ...

    images:
    - namePattern: gcr.io/projectsigstore/cosign*
      keys:
      - name: official-cosign-key
      secretRef:
        name: your-secret
        namespace: your-namespace

    ...
```

Policy Controller:

```
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  images:
  - glob: gcr.io/projectsigstore/cosign*
```

Where `POLICY-NAME` is the name of the cluster image policy you want to use.

Specifying policy mode

If `AllowUnmatchedImages` is set to `true` in the Image Policy Webhook deployment, create the following policy in the cluster:

```
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: allow-unmatched-image-policy
spec:
```

```
images:
- glob: "**"
  authorities:
  - static:
      action: pass
```

Configuring Supply Chain Security Tools - Policy

This topic describes how you can configure Supply Chain Security Tools - Policy. SCST - Policy requires extra configuration steps to verify your container images.

Admission of Images

An image is admitted after it is validated against a policy with matching image pattern, and where at least one valid signature is obtained from the authorities provided in a matched ClusterImagePolicy.

If more than one policy exists with a matching image pattern, *ALL* of the policies must have at least one passing authority for the image.

Including Namespaces

The Policy Controller only validates resources in namespaces that have chosen to opt-in. This is done by adding the label `policy.sigstore.dev/include: "true"` to the namespace resource.

```
kubectl label namespace my-secure-namespace policy.sigstore.dev/include=true
```


Caution

Without a Policy Controller ClusterImagePolicy applied, there are fallback behaviors where images are validated against the public Sigstore Rekor and Fulcio servers by using a keyless authority flow. Therefore, if the deploying image is signed publicly by a third-party using the keyless authority flow, the image is admitted as it can validate against the public Rekor and Fulcio. To avoid this behavior, develop, and apply a ClusterImagePolicy that applies to the images being deployed in the namespace.

Create a `ClusterImagePolicy` resource

The cluster image policy is a custom resource containing the following properties:

`images`

In a ClusterImagePolicy, `spec.images` specifies a list of glob matching patterns. These patterns are matched against the image digest in `PodSpec` for resources attempting deployment.

Policy Controller defines the following globs by default:

- If `*` is specified, the `glob` matching behavior is `index.docker.io/library/*`.
- If `*/*` is specified, the `glob` matching behavior is `index.docker.io/*/*`.

With these defaults, you require the `glob` pattern `**` to match against all images. If your image is hosted on Docker Hub, include `index.docker.io` as the host for the glob.

A sample ClusterImagePolicy which matches against all images using glob:

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  images:
  - glob: "**"
```

mode

In a ClusterImagePolicy, `spec.mode` specifies the action of a policy:

- `enforce`: The default behavior. If the policy fails to validate the image, the policy fails.
- `warn`: If the policy fails to validate the image, validation error messages are converted to warnings and the policy passes.

A sample of a ClusterImagePolicy which has `warn` mode configured.

```
---
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  mode: warn
```

Where `POLICY-NAME` is the name of the policy you want to configure your ClusterImagePolicy with.

When `enforce` mode is set, an image that fails validation is not admitted.

Sample output message:

```
error: failed to patch: admission webhook "policy.sigstore.dev" denied the request: va
lidation failed: failed policy: POLICY-NAME: spec.template.spec.containers[0].image
IMAGE-REFERENCE signature key validation failed for authority authority-0 for IMAGE-RE
FERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
failed policy: POLICY-NAME: spec.template.spec.containers[1].image
IMAGE-REFERENCE signature key validation failed for authority authority-0 for IMAGE-RE
FERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
```

When `warn` mode is set, an image that fails validation is admitted.

Sample output message:

```
Warning: failed policy: POLICY-NAME: spec.template.spec.containers[0].image
Warning: IMAGE-REFERENCE signature key validation failed for authority authority-0 for
IMAGE-REFERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
Warning: failed policy: POLICY-NAME: spec.template.spec.containers[1].image
Warning: IMAGE-REFERENCE signature key validation failed for authority authority-0 for
IMAGE-REFERENCE: GET IMAGE-SIGNATURE-REFERENCE: DENIED: denied; denied
```

If you don't want a `Warning` output message, you can configure a `static.action pass` authority to allow expected unsigned images. For example, you may want to allow unsigned images if your policy controller runs on a development environment, and you need to iterate quickly. For information about static action authorities, see Static Action.

match

You can use `match` to filter resources using group, version, kind, or labels in a selected namespace to enforce the defined policy. If the list of matching resources is empty, all core resources are used by default.

For example, you can filter all `v1 cronjobs` with the label `app: tap` in a namespace that is labeled for policy enforcement:

```
spec:
  match:
  - group: batch
    resource: cronjobs
    version: v1
    selector:
      matchLabels:
        app: tap
```

authorities

Authorities listed in the `authorities` block of the ClusterImagePolicy are `key` or `keyless` specifications.

key

Each `key` authority can contain a PEM-encoded ECDSA public key, a `secretRef`, or a `kms` path.

The policy resyncs with KMS referenced every 10 hours. Any updates to the secret in KMS is pulled in during the refresh. To force a resync, the policy must be deleted and recreated.

Important

Only ECDSA public keys are supported.

```
spec:
  authorities:
    - key:
        data: |
          -----BEGIN PUBLIC KEY-----
          ...
          -----END PUBLIC KEY-----
    - key:
        secretRef:
          name: secretName
    - key:
        kms: KMSPATH
```

Where `KMSPATH` is the name of the KMS path you want to configure in your key authority.

Note

The secret referenced in `key.secretRef.name` must be created in the `cosign-system` namespace or the namespace where the Policy Controller is installed. This secret must only contain one `data` entry with the public key.

keyless

Note

Keyless support is deactivated by default. See Install Supply Chain Security Tools - Policy Controller.

Each keyless authority can contain a Fulcio URL, a Rekor URL, a certificate, or an array of identities.

Identities are represented with a combination of `issuer` or `issuerRegExp` with `subject` or `subjectRegExp`.

- `issuer`: Defines the issuer for this identity.
- `issuerRegExp`: Specifies a regular expression to match the issuer for this identity.
- `subject`: Defines the subject for this identity.
- `subjectRegExp`: Specifies a regular expression to match the subject for this identity.

An example of keyless authority structure:

```
spec:
  authorities:
    - keyless:
        url: https://fulcio.example.com
        ca-cert:
          data: Certificate Data
        identities:
          - issuer: https://accounts.google.com
            subjectRegExp: .*@example.com
          - issuer: https://token.actions.githubusercontent.com
            subject: https://github.com/mycompany/*/.github/workflows/*@*
      ctlog:
        url: https://rekor.example.com
    - keyless:
        url: https://fulcio.example.com
        ca-cert:
          secretRef:
            name: secretName
        identities:
          - issuerRegExp: .*kubernetes.default.*
            subjectRegExp: .*kubernetes.io/namespaces/default/serviceaccounts/default
```

The authorities are evaluated using the `any of` operator to admit container images. For each pod, the Policy Controller iterates over the list of containers and init containers. For every policy that matches against the images, they must each have at least one valid signature obtained using the authorities specified. If an image does not match any policy, the Policy Controller does not admit the image.

`static.action`

ClusterImagePolicy authorities are configured to always `pass` or `fail` with `static.action`.

Sample `ClusterImagePolicy` with static action `fail`.

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: POLICY-NAME
spec:
  authorities:
  - static:
      action: fail
```

Where `POLICY-NAME` is the name of the policy you want to configure your ClusterImagePolicy with.

A sample output of static action `fail`:

```
error: failed to patch: admission webhook "policy.sigstore.dev" denied the request: va
lidation failed: failed policy: POLICY-NAME: spec.template.spec.containers[0].image
IMAGE-REFERENCE disallowed by static policy
```

```
failed policy: POLICY-NAME: spec.template.spec.containers[1].image
IMAGE-REFERENCE disallowed by static policy
```

Images that are unsigned in a namespace with validation enabled are admitted with an authority with static action `pass`.

This applies when you are configuring a policy with `static.action pass` for `tap-packages` images. Another policy is then configured to validate signed images produced by Tanzu Build Service. This allows images from `tap-packages`, which are unsigned and required by the platform, to be admitted while still validating signed built images from Tanzu Build Service. See Configure your supply chain to sign and verify your image builds.

If `Warning` messages are desirable for admitted images where validation failed, you can configure a policy with `warn` mode and valid authorities. For information about ClusterImagePolicy modes, see Mode.

Provide credentials for the package

There are three ways the package reads credentials to authenticate to registries protected by authentication:

1. Reading `imagePullSecrets` directly from the resource being admitted. See Container image pull secrets in the Kubernetes documentation.
2. Reading `imagePullSecrets` from the service account the resource is running as. See Arranging for imagePullSecrets to be automatically attached in the Kubernetes documentation.
3. Reading a `secretRef` from the `ClusterImagePolicy` resource's `signaturePullSecrets` when specifying the cosign signature source.

Authentication can fail for the following scenarios:

- A not valid credential is specified in the `imagePullSecrets` of the resource or in the service account the resource runs as.
- A not valid credential is specified in the `ClusterImagePolicy signaturePullSecrets` text box.

Provide secrets for authentication in your policy

You can provide secrets for authentication as part of the policy configuration. The `oci` location is the image location or a remote location where signatures are configured to be stored during signing. The `signaturePullSecrets` is available in the `cosign-system` namespace or the namespace where the Policy Controller is installed.

By default, `imagePullSecrets` from the resource or service account is used while the default `oci` location is the image location.

See the following example:

```
spec:
  authorities:
    - key:
        data: |
          -----BEGIN PUBLIC KEY-----
          ...
          -----END PUBLIC KEY-----
      source:
        - oci: registry.example.com/project/signature-location
          signaturePullSecrets:
            - name: MY-SECRET
```

```
    - keyless:
        url: https://fulcio.example.com
      source:
        - oci: registry.example.com/project/signature-location
          signaturePullSecrets:
            - name: MY-SECRET
```

Where `MY-SECRET` is the name of the secret you want to use with your credentials.

VMware recommends using a set of credentials with the least amount of privilege that allows reading the signature stored in your registry.

Verify your configuration

A sample policy:

```
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: image-policy
spec:
  images:
  - glob: "gcr.io/projectsigstore/cosign*"
  authorities:
  - name: official-cosign-key
    key:
      data: |
        -----BEGIN PUBLIC KEY-----
        MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhyQCx0E9wQWSFI9ULGwy3BuRklnt
        IqozONbbdbqz11hlRJy9c7SG+hdcFl9jE9uE/dwtuwU2MqU9T/cN0YkWww==
        -----END PUBLIC KEY-----
```

When using the sample policy, run these commands to verify your configuration:

1. Verify that the Policy Controller admits the signed image that validates with the configured public key. Run:

   ```
   kubectl run cosign \
     --image=gcr.io/projectsigstore/cosign:v1.2.1 \
     --dry-run=server
   ```

   For example:

   ```
   $ kubectl run cosign \
     --image=gcr.io/projectsigstore/cosign:v1.2.1 \
     --dry-run=server
   pod/cosign created (server dry run)
   ```

2. Verify that the Policy Controller rejects the unmatched image. Run:

   ```
   kubectl run busybox --image=busybox --dry-run=server
   ```

   For example:

   ```
   $ kubectl run busybox --image=busybox --dry-run=server
     Error from server (BadRequest): admission webhook "policy.sigstore.dev" denie
   d the request: validation failed: no matching policies: spec.containers[0].imag
   e
     index.docker.io/library/busybox@sha256:3614ca5eacf0a3a1bcc361c939202a974b4902
   b9334ff36eb29ffe9011aaad83
   ```

In the output, it did not specify which authorities were used as there was no policy found that matched the image. Therefore, the image fails to validate for a signature and fails to deploy.

3. Verify that the Policy Controller rejects a matched image signed with a different key than the one configured. Run:

```
kubectl run cosign-fail \
  --image=gcr.io/projectsigstore/cosign:v0.3.0 \
  --dry-run=server
```

For example:

```
$ kubectl run cosign-fail \
    --image=gcr.io/projectsigstore/cosign:v0.3.0 \
    --dry-run=server
  Error from server (BadRequest): admission webhook "policy.sigstore.dev" denie
d the request: validation failed: failed policy: image-policy: spec.containers
[0].image
  gcr.io/projectsigstore/cosign@sha256:135d8c5e27bdc917f04b415fc947d7d5b1137f99
bb8fa00bffc3eca1856e9c52 failed to validate public keys with authority official
-cosign-key for gcr.io/projectsigstore/cosign@sha256:135d8c5e27bdc917f04b415fc9
47d7d5b1137f99bb8fa00bffc3eca1856e9c52: no matching signatures:
```

In the output, it specifies which authorities were used for validation when a policy was found that matched the image. In this case, the authority used was `official-cosign-key`. If no name is specified, it is defaulted to `authority-#`.

Overview of Supply Chain Security Tools - Scan

This topic gives you an overview of use cases, features, and CVEs for Supply Chain Security Tools (SCST) - Scan.

Overview

With Supply Chain Security Tools - Scan, you can build and deploy secure, trusted software that complies with your corporate security requirements. Supply Chain Security Tools - Scan provides scanning and gatekeeping capabilities that Application and DevSecOps teams can incorporate early in their path to production as it is a known industry best practice for reducing security risk and ensuring more efficient remediation.

Language support

For information about the languages and frameworks that are supported by Tanzu Application Platform components, see the Language and framework support in Tanzu Application Platform table.

Use cases

The following use cases apply to Supply Chain Security Tools - Scan:

- Use your scanner as a plug-in to scan source code repositories and images for known Common Vulnerabilities and Exposures (CVEs) before deploying to a cluster.
- Identify CVEs by continuously scanning each new code commit or each new image built.
- Analyze scan results against user-defined policies by using Open Policy Agent.

- Produce vulnerability scan results and post them to the Supply Chain Security Tools - Store from where they are queried.

Supply Chain Security Tools - Scan features

The following Supply Chain Security Tools - Scan features enable the Use cases:

- Kubernetes controllers to run scan TaskRuns.
- Custom Resource Definitions (`CRD`s) for Image and Source Scan.
- `CRD` for a scanner plug-in. Example is available by using Anchore's Syft and Grype.
- `CRD` for policy enforcement.
- Enhanced scanning coverage by analyzing the Cloud Native Buildpack SBoMs that Tanzu Build Service images provide.

A Note on Vulnerability Scanners

Although vulnerability scanning is an important practice in DevSecOps and the benefits of it are widely recognized and accepted, it is important to remember that there are limitations present that impact its efficacy. The following examples illustrate the limitations that are prevalent in most scanners today:

Missed CVEs

One limitation of all vulnerability scanners is that there is no one tool that can find 100% of all CVEs, which means there is always a risk that a missed CVE can be exploited. Some reasons for missed CVEs include:

- The scanner does not detect the vulnerability because it is just discovered and the CVE databases that the scanner checks against are not updated yet.
- Scanners verify different CVE sources based on the detected package type and OS.
- The scanner might not fully support a particular programming language, packaging system or manifest format.
- The scanner might not implement binary analysis or fingerprinting.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.
- When vendors register impacted software with NVD, the provided information might not exactly match the values in the release artifacts.

False positives

Vulnerability scanners cannot always access the information to accurately identify whether a CVE exists. This often leads to an influx of false positives where the tool mistakenly flags something as a vulnerability when it isn't. Unless a user is specialized in security or is deeply familiar with what is deemed to be a vulnerable component by the scanner, assessing and determining false positives becomes a challenging and time-consuming activity. Some reasons for a false positive flag include:

- A component might be misidentified due to similar names.
- A subcomponent might be identified as the parent component.
- A component is correctly identified but the impacted function is not on a reachable code path.

- A component's impacted function is on a reachable code path but is not a concern due to the specific environment or configuration.
- The version of a component might be incorrectly flagged as impacted.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.

So what can you do to protect yourselves and your software?

Although vulnerability scanning is not a perfect solution, it is an essential part of the process for keeping your organization secure. You can take the following measures to maximize the benefits while minimizing the impact of the limitations:

- Scan more continuously and comprehensively to identify and remediate zero-day vulnerabilities quicker. Comprehensive scanning can be achieved by:
    - scanning earlier in the development cycle to ensure issues can be addressed more efficiently and do not delay a release. Tanzu Application Platform includes security practices such as source and container image vulnerability scanning earlier in the path to production for application teams.
    - scanning any base images in use. Tanzu Application Platform image scanning includes the ability to recognize and scan the OS packages from a base image.
    - scanning running software in test, stage, and production environments at a regular cadence.
    - generating accurate provenance at any level so that scanners have a complete picture of the dependencies to scan. This is where a software bill of materials (SBoM) comes into play. To help you automate this process, VMware Tanzu Build Service, leveraging Cloud Native Buildpacks, generates an SBoM for buildpack-based projects. Since this SBoM is generated during the image building stage, it is more accurate and complete than one generated earlier or later in the release life cycle. This is because it can highlight dependencies introduced at the time of build that might introduce potential for compromise.
- Scan by using multiple scanners to maximize CVE coverage.
- Practice keeping your dependencies up-to-date.
- Reduce overall surface area of attack by:
    - using smaller dependencies.
    - reducing the amount of third party dependencies when possible.
    - using distroless base images when possible.
- Maintain a central record of false positives to ease CVE triaging and remediation efforts.

Overview of Supply Chain Security Tools - Scan

This topic gives you an overview of use cases, features, and CVEs for Supply Chain Security Tools (SCST) - Scan.

Overview

With Supply Chain Security Tools - Scan, you can build and deploy secure, trusted software that complies with your corporate security requirements. Supply Chain Security Tools - Scan provides scanning and gatekeeping capabilities that Application and DevSecOps teams can incorporate early in their path to production as it is a known industry best practice for reducing security risk and ensuring more efficient remediation.

Language support

For information about the languages and frameworks that are supported by Tanzu Application Platform components, see the Language and framework support in Tanzu Application Platform table.

Use cases

The following use cases apply to Supply Chain Security Tools - Scan:

- Use your scanner as a plug-in to scan source code repositories and images for known Common Vulnerabilities and Exposures (CVEs) before deploying to a cluster.
- Identify CVEs by continuously scanning each new code commit or each new image built.
- Analyze scan results against user-defined policies by using Open Policy Agent.
- Produce vulnerability scan results and post them to the Supply Chain Security Tools - Store from where they are queried.

Supply Chain Security Tools - Scan features

The following Supply Chain Security Tools - Scan features enable the Use cases:

- Kubernetes controllers to run scan TaskRuns.
- Custom Resource Definitions (`CRD`s) for Image and Source Scan.
- `CRD` for a scanner plug-in. Example is available by using Anchore's Syft and Grype.
- `CRD` for policy enforcement.
- Enhanced scanning coverage by analyzing the Cloud Native Buildpack SBoMs that Tanzu Build Service images provide.

A Note on Vulnerability Scanners

Although vulnerability scanning is an important practice in DevSecOps and the benefits of it are widely recognized and accepted, it is important to remember that there are limitations present that impact its efficacy. The following examples illustrate the limitations that are prevalent in most scanners today:

Missed CVEs

One limitation of all vulnerability scanners is that there is no one tool that can find 100% of all CVEs, which means there is always a risk that a missed CVE can be exploited. Some reasons for missed CVEs include:

- The scanner does not detect the vulnerability because it is just discovered and the CVE databases that the scanner checks against are not updated yet.
- Scanners verify different CVE sources based on the detected package type and OS.
- The scanner might not fully support a particular programming language, packaging system or manifest format.
- The scanner might not implement binary analysis or fingerprinting.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.

- When vendors register impacted software with NVD, the provided information might not exactly match the values in the release artifacts.

False positives

Vulnerability scanners cannot always access the information to accurately identify whether a CVE exists. This often leads to an influx of false positives where the tool mistakenly flags something as a vulnerability when it isn't. Unless a user is specialized in security or is deeply familiar with what is deemed to be a vulnerable component by the scanner, assessing and determining false positives becomes a challenging and time-consuming activity. Some reasons for a false positive flag include:

- A component might be misidentified due to similar names.
- A subcomponent might be identified as the parent component.
- A component is correctly identified but the impacted function is not on a reachable code path.
- A component's impacted function is on a reachable code path but is not a concern due to the specific environment or configuration.
- The version of a component might be incorrectly flagged as impacted.
- The detected component does not always include a canonical name and vendor, requiring the scanner to infer and attempt fuzzy matching.

So what can you do to protect yourselves and your software?

Although vulnerability scanning is not a perfect solution, it is an essential part of the process for keeping your organization secure. You can take the following measures to maximize the benefits while minimizing the impact of the limitations:

- Scan more continuously and comprehensively to identify and remediate zero-day vulnerabilities quicker. Comprehensive scanning can be achieved by:
    - scanning earlier in the development cycle to ensure issues can be addressed more efficiently and do not delay a release. Tanzu Application Platform includes security practices such as source and container image vulnerability scanning earlier in the path to production for application teams.
    - scanning any base images in use. Tanzu Application Platform image scanning includes the ability to recognize and scan the OS packages from a base image.
    - scanning running software in test, stage, and production environments at a regular cadence.
    - generating accurate provenance at any level so that scanners have a complete picture of the dependencies to scan. This is where a software bill of materials (SBoM) comes into play. To help you automate this process, VMware Tanzu Build Service, leveraging Cloud Native Buildpacks, generates an SBoM for buildpack-based projects. Since this SBoM is generated during the image building stage, it is more accurate and complete than one generated earlier or later in the release life cycle. This is because it can highlight dependencies introduced at the time of build that might introduce potential for compromise.
- Scan by using multiple scanners to maximize CVE coverage.
- Practice keeping your dependencies up-to-date.
- Reduce overall surface area of attack by:
    - using smaller dependencies.
    - reducing the amount of third party dependencies when possible.

    - using distroless base images when possible.
- Maintain a central record of false positives to ease CVE triaging and remediation efforts.

Install Supply Chain Security Tools - Scan

This topic describes how you can install Supply Chain Security Tools - Scan from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install SCST - Scan. For information about profiles, see Components and installation profiles.

Prerequisites

Before installing SCST - Scan:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Supply Chain Security Tools - Store for scan results to persist. The integration with SCST - Store are handled in:
    - **Single Cluster:** The SCST - Store is present in the same cluster where SCST - Scan and the `ScanTemplates` are present.
    - **Multi-Cluster:** The SCST - Store is present in a different cluster (e.g.: view cluster) where the SCST - Scan and `ScanTemplates` are present.
    - **Integration Deactivated:** The SCST - Scan deployment is not required to communicate with SCST - Store.

  For information about SCST - Store, see Using the Supply Chain Security Tools - Store.

Configure properties

When you install the SCST - Scan (Scan controller), you can configure the following optional properties:

Key	Default	Type	Description	ScanTemplate Version
resources.limits.cpu	250m	integer/string	Limits describes the maximum amount of CPU resources allowed.	*n/a*
resources.limits.memory	256Mi	integer/string	Limits describes the maximum amount of memory resources allowed.	*n/a*
resources.requests.cpu	100m	integer/string	Requests describes the minimum amount of CPU resources required.	*n/a*
resources.requests.memory	128Mi	integer/string	Requests describes the minimum amount of memory resources required.	*n/a*
namespace	scan-link-system	string	Deployment namespace for the Scan Controller	*n/a*
metadataStore.caSecret.importFromNamespace	metadata-store	string	Namespace from which you import the Insight Metadata Store CA Cert	earlier than v1.2.0

Key	Default	Type	Description	ScanTemplate Version
metadataStore.caSecret.name	app-tls-cert	string	Name of deployed secret with key ca.crt holding the CA Cert of the Insight Metadata Store	earlier than v1.2.0
metadataStore.clusterRole	metadata-store-read-write	string	Name of the deployed ClusterRole for read/write access to the Insight Metadata Store deployed in the same cluster	earlier than v1.2.0
metadataStore.url	https://metadata-store-app.metadata-store.svc.cluster.local:8443	string	URL of the Insight Metadata Store	earlier than v1.2.0
metadataStore.authSecret.importFromNamespace	*n/a*	string	Namespace from which to import the Insight Metadata Store auth_token	earlier than v1.2.0
metadataStore.authSecret.name	*n/a*	string	Name of deployed secret with key auth_token	earlier than v1.2.0
retryScanJobsSecondsAfterError	60	integer	Seconds to wait before retrying errored scans	v1.3.1 and later
caCertData	""	string	The custom certificates trusted by the scans' connections	v1.4.0 and later
certIssuer	""	string	The common certificate issuer for the cluster	v1.5.0 and later
controller.pullSecret	"controller-secret-ref"	string	Reference to the secret used for pulling the controller image from private registry. Set to empty if deploying from a public registry.	v1.5.0 and later
docker.import	true	Boolean	Import `controller.pullSecret` from another namespace (requires secretgen-controller). Set to false if the secret is present.	v1.5.0 and later
kubeRbacProxy.certRef	""	string	Reference to the secret which holds certificate for kube-rbac-proxy. The Certificate enables secure connection to the metric proxy.	v1.5.0 and later
kubeRbacProxy.tls.minVersion	""	string	Minimum TLS version supported by kube-rbac-proxy. Value must match version names from https://golang.org/pkg/crypto/tls/#pkg-constants.	v1.5.0 and later
kubeRbacProxy.tls.ciphers	empty array	array of strings	Comma-separated list of cipher suites for the server supported by kube-rbac-proxy. Values are from tls package constants (https://golang.org/pkg/crypto/tls/#pkg-constants).	v1.5.0 and later

When you install the SCST - Scan (Grype scanner), you can configure the following optional properties:

Key	Default	Type	Description	ScanTemplate Version
resources.requests.cpu	250m	integer/string	Requests describes the minimum amount of CPU resources required.	

Key	Default	Type	Description	ScanTemplate Version
resources.requests.memory	128Mi	integer/string	Requests describes the minimum amount of memory resources required.	
scanner.serviceAccount	grype-scanner	string	Name of scan pod's service ServiceAccount	
scanner.serviceAccountAnnotations	nil	object	Annotations added to ServiceAccount	
targetImagePullSecret	*n/a*	string	Reference to the secret used for pulling images from private registry	
targetSourceSshSecret	*n/a*	string	Reference to the secret containing SSH credentials for cloning private repositories	
namespace	default	string	Deployment namespace for the Scan Templates	*n/a*
metadataStore.url	https://metadata-store-app.metadata-store.svc.cluster.local:8443	string	URL of the Insight Metadata Store	v1.2.0 and earlier
metadataStore.authSecret.name	*n/a*	string	Name of deployed secret with key auth_token	v1.2.0 and earlier
metadataStore.authSecret.importFromNamespace	*n/a*	string	Namespace from which to import the Insight Metadata Store auth_token	v1.2.0 and earlier
metadataStore.caSecret.importFromNamespace	metadata-store	string	Namespace from which to import the Insight Metadata Store CA Cert	v1.2.0 and earlier
metadataStore.caSecret.name	app-tls-cert	string	Name of deployed secret with key ca.crt holding the CA Cert of the Insight Metadata Store	v1.2.0 and earlier
metadataStore.clusterRole	metadata-store-read-write	string	Name of the deployed ClusterRole for read/write access to the Insight Metadata Store deployed in the same cluster	v1.2.0

Install

There are two options for installing Supply Chain Security Tools – Scan

Option 1: Install to multiple namespaces with the Namespace Provisioner

The Namespace Provisioner enables operators to securely automate the provisioning of multiple developer namespaces in a shared cluster. To install Supply Chain Security Tools – Scan by using the Namespace Provisioner, see Namespace Provisioner.

The Namespace Provisioner can also create scan policies across multiple developer namespaces. See Customize installation in the Namespace Provisioner documentation for configuration steps.

Option 2: Install manually to each individual namespace

The installation for Supply Chain Security Tools – Scan involves installing two packages:

- Scan controller
- Grype scanner

The Scan controller enables you to use a scanner, in this case, the Grype scanner. Ensure that both the Grype scanner and the Scan controller are installed.

To install SCST - Scan (Scan controller):

1. List version information for the package by running:

   ```
   tanzu package available list scanning.apps.tanzu.vmware.com --namespace tap-ins
   tall
   ```

   For example:

   ```
   $ tanzu package available list scanning.apps.tanzu.vmware.com --namespace tap-i
   nstall
   / Retrieving package versions for scanning.apps.tanzu.vmware.com...
     NAME                            VERSION       RELEASED-AT
     scanning.apps.tanzu.vmware.com   1.1.0
   ```

2. (Optional) Make changes to the default installation settings:

   If you are using Grype Scanner `v1.5.1 and later` or other supported scanners included with Tanzu Application Platform `v1.5.1 and later`, and do not want to use the default SCST - Store integration, deactivate the integration by appending the following field to the `values.yaml` file:

   ```
   ---
   metadataStore:
     url: "" # Deactivate Supply Chain Security Tools - Store integration
   ```

   If you are using Grype Scanner `v1.5.0` or other supported scanners included with Tanzu Application Platform `v1.5.0`, and do not want to use the default SCST - Store integration, deactivate the integration by appending the following field to the `values.yaml` file:

   ```
   ---
   metadataStore: {} # Deactivate Supply Chain Security Tools - Store integration
   ```

   If you are using Grype Scanner `v1.2.0 and earlier`, or the Snyk Scanner, the following scanning configuration might deactivate the embedded SCST - Store integration with a `scan-values.yaml` file.

   ```
   ---
   metadataStore:
     url: "" # Deactivate Supply Chain Security Tools - Store integration
   ```

   If your Grype Scanner version is earlier than v1.2.0, the scanning configuration must configure the store parameters. See v1.1 Install Supply Chain Security Tools - Scan.

   Run to retrieve other configurable settings and append the key-value pair to the previous `scan-values.yaml` file:

   ```
   tanzu package available get scanning.apps.tanzu.vmware.com/VERSION --values-sch
   ema -n tap-install
   ```

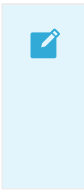
   Where `VERSION` is your package version number. For example, `1.1.0`.

3. Install the package by running:

```
tanzu package install scan-controller \
  --package scanning.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file scan-values.yaml
```

Where `VERSION` is your package version number. For example, `1.1.0`.

To install SCST - Scan (Grype scanner):



Note

To install Grype in multiple namespaces, use a namespace provisioner. See Namespace Provisioner.

1. List version information for the package by running:

   ```
   tanzu package available list grype.scanning.apps.tanzu.vmware.com --namespace t
   ap-install
   ```

   For example:

   ```
   $ tanzu package available list grype.scanning.apps.tanzu.vmware.com --namespace
   tap-install
   / Retrieving package versions for grype.scanning.apps.tanzu.vmware.com...
     NAME                                VERSION       RELEASED-AT
     grype.scanning.apps.tanzu.vmware.com  1.1.0
   ```

2. (Optional) Make changes to the default installation settings:

   To define the configuration for the SCST - Store integration in the `grype-values.yaml` file for the Grype Scanner:

   ```
   ---
   namespace: "DEV-NAMESPACE" # The developer namespace where the ScanTemplates ar
   e going to be deployed
   metadataStore:
     url: "METADATA-STORE-URL" # The base URL where the Store deployment can be re
   ached
     caSecret:
       name: "CA-SECRET-NAME" # The name of the secret containing the ca.crt
       importFromNamespace: "SECRET-NAMESPACE" # The namespace where Store is depl
   oyed (if single cluster) or where the connection secrets were created (if multi
   -cluster)
     authSecret:
       name: "TOKEN-SECRET-NAME" # The name of the secret containing the auth toke
   n to connect to Store
       importFromNamespace: "SECRET-NAMESPACE" # The namespace where the connectio
   n secrets were created (if multi-cluster)
   ```

   Note In a single cluster, the connection between the scanning pod and the metadata store happens inside the cluster and does not pass through ingress. This is automatically configured. You do not need to provide an ingress connection to the store. For information about troubleshooting issues with scanner to metadata store connection configuration, see Troubleshooting Scanner to MetadataStore Configuration.

Important

You must either define both the `METADATA-STORE-URL` and `CA-SECRET-NAME`, or not define them as they depend on each other.

Where:

- `DEV-NAMESPACE` is the namespace where you want to deploy the `ScanTemplates`. This is the namespace where the scanning feature runs.
- `METADATA-STORE-URL` is the base URL where the Supply Chain Security Tools (SCST) - Store deployment is reached, for example, `https://metadata-store-app.metadata-store.svc.cluster.local:8443`.
- `CA-SECRET-NAME` is the name of the secret containing the ca.crt to connect to the SCST - Store deployment.
- `SECRET-NAMESPACE` is the namespace where SCST - Store is deployed, if you are using a single cluster. If you are using multicluster, it is where the connection secrets were created.
- `TOKEN-SECRET-NAME` is the name of the secret containing the authentication token to connect to the SCST - Store deployment when installed in a different cluster, if you are using multicluster. If built images are pushed to the same registry as the Tanzu Application Platform images, this can reuse the `tap-registry` secret created in Add the Tanzu Application Platform package repository as described earlier.

Run to retrieve other configurable settings and append the key-value pair to the previous `grype-values.yaml` file:

```
tanzu package available get grype.scanning.apps.tanzu.vmware.com/VERSION --valu
es-schema -n tap-install
```

Where `VERSION` is your package version number. For example, `1.1.0`.

For example:

```
$ tanzu package available get grype.scanning.apps.tanzu.vmware.com/1.1.0 --valu
es-schema -n tap-install
| Retrieving package details for grype.scanning.apps.tanzu.vmware.com/1.1.0...
  KEY                        DEFAULT  TYPE    DESCRIPTION
  namespace                  default  string  Deployment namespace for the Scan
Templates
  resources.limits.cpu       1000m    <nil>   Limits describes the maximum amou
nt of cpu resources allowed.
  resources.requests.cpu     250m     <nil>   Requests describes the minimum am
ount of cpu resources required.
  resources.requests.memory  128Mi    <nil>   Requests describes the minimum am
ount of memory resources required.
  targetImagePullSecret      <EMPTY>  string  Reference to the secret used for
pulling images from private registry.
  targetSourceSshSecret      <EMPTY>  string  Reference to the secret containin
g SSH credentials for cloning private repositories.
```

Important

If `targetSourceSshSecret` is not set, the private source scan template is not installed.

3. Install the package by running:

```
tanzu package install grype-scanner \
  --package grype.scanning.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file grype-values.yaml
```

Where `VERSION` is your package version number. For example, `1.1.0`.

For example:

```
$ tanzu package install grype-scanner \
  --package grype.scanning.apps.tanzu.vmware.com \
  --version 1.1.0 \
  --namespace tap-install \
  --values-file grype-values.yaml
/ Installing package 'grype.scanning.apps.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'grype.scanning.apps.tanzu.vmware.com'
| Creating service account 'grype-scanner-tap-install-sa'
| Creating cluster admin role 'grype-scanner-tap-install-cluster-role'
| Creating cluster role binding 'grype-scanner-tap-install-cluster-rolebinding'
/ Creating package resource
- Package install status: Reconciling

 Added installed package 'grype-scanner' in namespace 'tap-install'
```

Upgrade Supply Chain Security Tools - Scan

This topic describes how you can upgrade Supply Chain Security Tools - Scan from the Tanzu Application Platform package repository.

You can perform a fresh install of SCST - Scan by following the instructions in Install Supply Chain Security Tools - Scan.

Prerequisites

Before you upgrade SCST - Scan, upgrade the Tanzu Application Platform by following the instructions in Upgrading Tanzu Application Platform.

General Upgrades for SCST - Scan

When you're upgrading to any version of SCST - Scan these are some factors to accomplish this task:

1. Inspect the Release Notes for the version you're upgrading to. There you can find any breaking changes for the installation.
2. Get the values schema for the package version you're upgrading to by running:

   ```
   tanzu package available get scanning.apps.tanzu.vmware.com/$VERSION --values-sc
   hema -n tap-install
   ```

   Where `$VERSION` is the new version. This gives you insights on the values you can configure in your `tap-values.yaml` for the new version.

Upgrading a scanner in all namespaces

This section describes how to upgrade a supported scanner in all namespaces. The procedure is different depending on the installation method:

1. Installation by using Namespace Provisioner
2. Manual installation

Installation by using Namespace Provisioner

All scanners installed by the Namespace Provisioner in all managed namespaces are upgraded automatically. For example, if you upgrade your installation of Tanzu Application Platform and the version of Grype is updated, all Grype scanners installed by the Namespace Provisioner for all managed namespaces are automatically upgraded.

Manual installation

1. If a scanner, such as Grype Scanner, was installed as part of Tanzu Application Platform by using the full profile, run to upgrade:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v VERSION --values-file tap-values.yaml -n tap-install
   ```

   Where `VERSION` is your Tanzu Application Platform version.

2. If a scanner, such as Grype Scanner, was installed by using component installation you must manually run:

   ```
   tanzu package installed update grype -p grype.scanning.apps.tanzu.vmware.com -v GRYPE-VERSION --values-file grype-values.yaml -n NAMESPACE
   ```

   Where:

   - `GRYPE-VERSION` is the version of Grype that you are upgrading to.
   - `NAMESPACE` is the namespace in which Grype is installed in.

Upgrade to Version v1.2.0

To upgrade from a previous version of SCST - Scan to the version `v1.2.0`:

1. Change the `SecretExports` from SCST - Store.

   SCST - Scan needs information to connect to the SCST - Store deployment, you must change where these secrets are exported to enable the connection with the version `v1.2.0` of SCST - Scan.

   - **For a single cluster deployment:**

     1. Edit the `tap-values.yaml` file you used to deploy SCST - Store to export the CA certificate to your developer namespace.

        ```
        metadata_store:
            ns_for_export_app_cert: "DEV-NAMESPACE"
        ```

        Note

        The `ns_for_export_app_cert` supports one namespace at a time. If you have multiple namespaces you can replace this value with a `*`, but this exports the CA certificate to all namespaces. Consider whether this increased visibility presents a risk.

2. Update Tanzu Application Platform to apply the changes:

```
tanzu package installed update tap -f tap-values.yaml -n tap-insta
ll
```

- **For a multi-cluster deployment:**

  You must reapply the SecretExport by changing the `toNamespace: scan-link-system` to `toNamespace: DEV-NAMESPACE`:

```
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
spec:
  toNamespace: "DEV-NAMESPACE"
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-auth-token
  namespace: metadata-store-secrets
spec:
  toNamespace: "DEV-NAMESPACE"
```

2. Update your `tap-values.yaml` file.

   The installation of the SCST - Scan and the Grype scanner have some changes. The connection to the SCST - Store component have moved to the Grype scanner package. To deactivate the connection from the SCST - Scan, which is still present for backwards compatibility, but is deprecated and is removed in `v1.3.0`.

```
# Deactivate scan controller embedded Supply Chain Security Tools - Store integ
ration
scanning:
  metadataStore:
    url: ""

# Install Grype Scanner v1.2.0
grype:
  namespace: "DEV-NAMESPACE" # The developer namespace where the ScanTemplates
are going to be deployed
  metadataStore:
    url: "METADATA-STORE-URL" # The base URL where the Store deployment can be
reached
    caSecret:
      name: "CA-SECRET-NAME" # The name of the secret containing the ca.crt
      importFromNamespace: "SECRET-NAMESPACE" # The namespace where Store is de
ployed (if single cluster) or where the connection secrets were created (if mul
ti-cluster)
    authSecret:
      name: "TOKEN-SECRET-NAME" # The name of the secret containing the auth to
ken to connect to Store
      importFromNamespace: "SECRET-NAMESPACE" # The namespace where the connect
ion secrets were created (if multi-cluster)
```

   For more insights on how to install Grype, see Install Supply Chain Security Tools - Scan (Grype Scanner).

Note

> If a mix of Grype templates, such as earlier than v1.2.0 and v1.2.0 and later, are used, both `scanning` and `grype` must configure the parameters. The secret must also export to both scan-link-system and the developer namespace. Do this by exporting to `*` or by defining multiple secrets and exports. If Grype is installed to multiple namespaces there must be corresponding exports. See Install Supply Chain Security Tools - Scan (Grype Scanner).

3. Update Tanzu Application Platform to apply the changes:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

4. Update the `ScanPolicy` to include the latest structure changes for `v1.2.0`.

   To update to the latest valid Rego File in the `ScanPolicy`, Enforce compliance policy using Open Policy Agent. `v1.2.0` introduced some breaking changes in the Rego File structure used for the `ScanPolicies`. For more information, see the Release Notes.

5. Verify the upgrade.

   You can run any `ImageScan` or `SourceScan` in your `DEV-NAMESPACE` where the Grype Scanner was installed, and it finishes. Here is a sample you can try to run to detect if everything upgraded.

   1. Create the `verify-upgrade.yaml` file in your system with the following content:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible",
"UnknownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e
:= match.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := i
nput.bom.components.component[_] }
      some i
```

```
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e
:= comp.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := v
uln.ratings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-public-image-scan
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: public-image-scan-template
  scanPolicy: scan-policy
```

2. Deploy the resources:

```
kubectl apply -f verify-upgrade.yaml -n DEV-NAMESPACE
```

3. View the scan results:

```
kubectl describe imagescan sample-public-image-scan -n DEV-NAMESPACE
```

If it is successful, the `ImageScan` goes to the `Failed` phase and shows the results of the scan in the `Status`.

Install another scanner for Supply Chain Security Tools - Scan

This topic describes how you can install scanners to work with Supply Chain Security Tools - Scan from the Tanzu Application Platform package repository.

Follow the instructions in this topic to install a scanner other than the out of the box Grype Scanner with SCST - Scan.

Prerequisites

Before installing a new scanner, install Supply Chain Security Tools - Scan. It must be present on the same cluster. The prerequisites for Scan are also required.

Note

Different scanners may have different limits. See Supported Scanner Matrix for Supply Chain Security Tools - Scan.

Install

To install a new scanner, follow these steps:

1. Complete scanner specific prerequisites for the scanner you're trying to install. For example, creating an API token to connect to the scanner.

- Snyk Scanner (Beta) is available for image scanning.
- Carbon Black Scanner (Beta) is available for image scanning.

2. List the available packages to discover what scanners you can use by running:

```
tanzu package available list --namespace tap-install
```

For example:

```
$ tanzu package available list --namespace tap-install
/ Retrieving available packages...
  NAME                                                DISPLAY-NAME
SHORT-DESCRIPTION
  grype.scanning.apps.tanzu.vmware.com                Grype Scanner for Supply
Chain Security Tools - Scan                          Default scan templates using A
nchore Grype
  snyk.scanning.apps.tanzu.vmware.com                 Snyk for Supply Chain Se
curity Tools - Scan                                   Default scan templates using
Snyk
  carbonblack.scanning.apps.tanzu.vmware.com          Carbon Black Scanner for
Supply Chain Security Tools - Scan                   Default scan templates using C
arbon Black
```

3. List version information for the scanner package by running:

```
tanzu package available list SCANNER-NAME --namespace tap-install
```

For example:

```
$ tanzu package available list snyk.scanning.apps.tanzu.vmware.com --namespace
tap-install
/ Retrieving package versions for snyk.scanning.apps.tanzu.vmware.com...
  NAME                                VERSION           RELEASED-AT
  snyk.scanning.apps.tanzu.vmware.com   1.0.0-beta.2
```

4. (Optional) Confirm that the secret created in Step 1 for scanner specific prerequisites is created.

5. Create a `values.yaml` to apply custom configurations to the scanner:

> **Note**
>
> This step might be required for some scanners but optional for others.

To list the values you can configure for any scanner, run:

```
tanzu package available get SCANNER-NAME/VERSION --values-schema -n tap-install
```

Where:

- `SCANNER-NAME` is the name of the scanner package you retrieved earlier.
- `VERSION` is your package version number. For example, `snyk.scanning.apps.tanzu.vmware.com/1.0.0-beta.2`.

For example:

```
$ tanzu package available get snyk.scanning.apps.tanzu.vmware.com/1.0.0-beta.2
--values-schema -n tap-install

KEY                                          DEFAULT
```

```
TYPE    DESCRIPTION
metadataStore.authSecret.name
string  Name of deployed Secret with key auth_token
metadataStore.authSecret.importFromNamespace
string  Namespace from which to import the Insight Metadata Store auth_token
metadataStore.caSecret.importFromNamespace    metadata-store
string  Namespace from which to import the Insight Metadata Store CA Cert
metadataStore.caSecret.name                   app-tls-cert
string  Name of deployed Secret with key ca.crt holding the CA Cert of the Insi
ght Metadata Store
metadataStore.clusterRole                     metadata-store-read-write
string  Name of the deployed ClusterRole for read/write access to the Insight M
etadata Store deployed in the same cluster
metadataStore.url                             https://metadata-store-app.metada
ta-store.svc.cluster.local:8443  string  Url of the Insight Metadata Store
namespace                                     default
string  Deployment namespace for the Scan Templates
resources.requests.cpu                        250m
<nil>   Requests describes the minimum amount of cpu resources required.
resources.requests.memory                     128Mi
<nil>   Requests describes the minimum amount of memory resources required.
resources.limits.cpu                          1000m
<nil>   Limits describes the maximum amount of cpu resources allowed.
snyk.tokenSecret.name
string  Reference to the secret containing a Snyk API Token as snyk_token.
targetImagePullSecret
string  Reference to the secret used for pulling images from private registry.
```

6. Define the `--values-file` flag to customize the default configuration:

   The `values.yaml` file you created earlier is referenced with the `--values-file` flag when running your Tanzu install command:

   ```
   tanzu package install REFERENCE-NAME \
     --package SCANNER-NAME \
     --version VERSION \
     --namespace tap-install \
     --values-file PATH-TO-VALUES-YAML
   ```

   Where:

   - `REFERENCE-NAME` is the name referenced by the installed package. For example, `grype-scanner`, `snyk-scanner`.
   - `SCANNER-NAME` is the name of the scanner package you retrieved earlier. For example, `snyk.scanning.apps.tanzu.vmware.com`.
   - `VERSION` is your package version number. For example, `1.0.0-beta.2`.
   - `PATH-TO-VALUES-YAML` is the path that points to the `values.yaml` file created earlier.

   For example:

   ```
   $ tanzu package install snyk-scanner \
     --package snyk.scanning.apps.tanzu.vmware.com \
     --version 1.1.0 \
     --namespace tap-install \
     --values-file values.yaml
   / Installing package 'snyk.scanning.apps.tanzu.vmware.com'
   | Getting namespace 'tap-install'
   | Getting package metadata for 'snyk.scanning.apps.tanzu.vmware.com'
   | Creating service account 'snyk-scanner-tap-install-sa'
   | Creating cluster admin role 'snyk-scanner-tap-install-cluster-role'
   | Creating cluster role binding 'snyk-scanner-tap-install-cluster-rolebinding'
   / Creating package resource
   - Package install status: Reconciling
   ```

```
Added installed package 'snyk-scanner' in namespace 'tap-install'
```

Verify Installation

To verify the installation create an `ImageScan` or `SourceScan` referencing one of the newly added `ScanTemplates` for the scanner.

1. (Optional) Create a `ScanPolicy` formatted for the output specific to the scanner you are installing, to reference in the `ImageScan` or `SourceScan`.

   ```
   kubectl apply -n $DEV_NAMESPACE -f SCAN-POLICY-YAML
   ```

   Note

   As vulnerability scanners output different formats, the `ScanPolicies` can vary. For information about policies and samples, see Enforce compliance policy using Open Policy Agent.

2. Retrieve available `ScanTemplates` from the namespace where the scanner is installed:

   ```
   kubectl get scantemplates -n DEV-NAMESPACE
   ```

   Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

   For example:

   ```
   $ kubectl get scantemplates
   NAME                                AGE
   blob-source-scan-template           10d
   private-image-scan-template         10d
   public-image-scan-template          10d
   public-source-scan-template         10d
   snyk-private-image-scan-template    10d
   snyk-public-image-scan-template     10d
   ```

3. Create the following ImageScan YAML:

   Note

   Some scanners do not support both `ImageScan` and `SourceScan`.

   ```
   apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
   kind: ImageScan
   metadata:
     name: sample-scanner-public-image-scan
   spec:
     registry:
       image: "nginx:1.16"
     scanTemplate: SCAN-TEMPLATE
     scanPolicy: SCAN-POLICY # Optional
   ```

   Where:

   - `SCAN-TEMPLATE` is the name of the installed `ScanTemplate` in the `DEV-NAMESPACE` you retrieved earlier.

- `SCAN-POLICY` it's an optional reference to an existing `ScanPolicy` in the same `DEV-NAMESPACE`.

For example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-snyk-public-image-scan
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: snyk-public-image-scan-template
  scanPolicy: snyk-scan-policy
```

4. Create the following SourceScan YAML:

Note

Some scanners do not support both `ImageScan` and `SourceScan`.

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: sample-scanner-public-source-scan
spec:
  git:
    url: "https://github.com/houndci/hound.git"
    revision: "5805c650"
  scanTemplate: SCAN-TEMPLATE
  scanPolicy: SCAN-POLICY # Optional
```

Where:

- `SCAN-TEMPLATE` is the name of the installed `ScanTemplate` in the `DEV-NAMESPACE` you retrieved earlier.
- `SCAN-POLICY` is an optional reference to an existing `ScanPolicy` in the same `DEV-NAMESPACE`.

For example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: sample-grype-public-source-scan
spec:
  git:
    url: "https://github.com/houndci/hound.git"
    revision: "5805c650"
  scanTemplate: public-source-scan-template
  scanPolicy: scan-policy
```

5. Apply the ImageScan and SourceScan YAMLs:

To run the scans, apply them to the cluster by running these commands:

`ImageScan`:

```
kubectl apply -f PATH-TO-IMAGE-SCAN-YAML -n DEV-NAMESPACE
```

Where `PATH-TO-IMAGE-SCAN-YAML` is the path to the YAML file created earlier.

`SourceScan`:

```
kubectl apply -f PATH-TO-SOURCE-SCAN-YAML -n DEV-NAMESPACE
```

Where `PATH-TO-SOURCE-SCAN-YAML` is the path to the YAML file created earlier.

For example:

```
$ kubectl apply -f imagescan.yaml -n my-apps
imagescan.scanning.apps.tanzu.vmware.com/sample-snyk-public-image-scan created

$ kubectl apply -f sourcescan.yaml -n my-apps
sourcescan.scanning.apps.tanzu.vmware.com/sample-grype-public-source-scan creat
ed
```

6. To verify the integration, get the scan to see if it completed by running:

   For `ImageScan`:

   ```
   kubectl get imagescan IMAGE-SCAN-NAME -n DEV-NAMESPACE
   ```

   Where `IMAGE-SCAN-NAME` is the name of the `ImageScan` as defined in the YAML file created earlier.

   For `SourceScan`:

   ```
   kubectl get sourcescan SOURCE-SCAN-NAME -n DEV-NAMESPACE
   ```

   Where `SOURCE-SCAN-NAME` is the name of the `SourceScan` as defined in the YAML file created earlier.

   For example:

   ```
   $ kubectl get imagescan sample-snyk-public-image-scan -n my-apps
   NAME                            PHASE       SCANNEDIMAGE   AGE   CRITICAL   HIG
   H   MEDIUM   LOW   UNKNOWN   CVETOTAL
   sample-snyk-public-image-scan   Completed   nginx:1.16     26h   0          114
   58       314   0         486

   $ kubectl get sourcescan sample-grype-public-source-scan -n my-apps
   NAME                                                                    PHASE
   SCANNEDREVISION   SCANNEDREPOSITORY                       AGE     CRITICAL   HIG
   H   MEDIUM   LOW   UNKNOWN   CVETOTAL
   sourcescan.scanning.apps.tanzu.vmware.com/grypesourcescan-sample-public   Compl
   eted   5805c650          https://github.com/houndci/hound.git   8m34s   21
   121    112      9     0         263
   ```

   Note

   If you define a `ScanPolicy` for the scans and the evaluation finds a violation, the `Phase` is `Failed` instead of `Completed`. In both cases the scan finished.

7. Clean up:

   ```
   kubectl delete -f PATH-TO-SCAN-YAML -n DEV-NAMESPACE
   ```

   Where `PATH-TO-SCAN-YAML` is the path to the YAML file created earlier.

Install scanner to multiple namespaces

To install a Scanner to multiple namespaces, VMware recommends using a namespace provisioner. See Namespace Provisioner

Configure Tanzu Application Platform Supply Chain to use new scanner

In order to scan your images with the new scanner installed in the Out of the Box Supply Chain with Testing and Scanning, you must update your Tanzu Application Platform installation.

Add the `ootb_supply_chain_testing_scanning.scanning` section to your `tap-values.yaml` and perform a Tanzu Application Platform update.

You can define which `ScanTemplates` is used for both `SourceScan` and `ImageScan`. The default values are the Grype Scanner `ScanTemplates`, but they are overwritten by any other `ScanTemplate` present in your `DEV-NAMESPACE`. The same applies to the `ScanPolicies` applied to each kind of scan.

```
ootb_supply_chain_testing_scanning:
  scanning:
    image:
      template: IMAGE-SCAN-TEMPLATE
      policy: IMAGE-SCAN-POLICY
    source:
      template: SOURCE-SCAN-TEMPLATE
      policy: SOURCE-SCAN-POLICY
```

Note

For the Supply Chain to work properly, the `SOURCE-SCAN-TEMPLATE` must support blob files and the `IMAGE-SCAN-TEMPLATE` must support private images.

For example:

```
ootb_supply_chain_testing_scanning:
  scanning:
    image:
      template: snyk-private-image-scan-template
      policy: snyk-scan-policy
    source:
      template: blob-source-scan-template
      policy: scan-policy
```

Uninstall Scanner

To replace the scanner in the Supply Chain, follow the steps mentioned in Configure TAP Supply Chain to Use New Scanner. After the scanner is no longer required by the Supply Chain, you can remove the package by running:

```
tanzu package installed delete REFERENCE-NAME \
    --namespace tap-install
```

Where `REFERENCE-NAME` is the name you identified the package with, when installing in the Install section. For example, `grype-scanner`, `snyk-scanner`.

For example:

```
$ tanzu package installed delete snyk-scanner \
    --namespace tap-install
```

Other Available Scanner Integrations

In addition to providing the above supported integrations, VMware encourages the broader community to support VMware in our goal of integrating with customers' preferred CVE scanners.

Additional integrations:

- Prisma Scanner (Alpha) is available for source and image scanning.
- Trivy Scanner (Alpha) is available for source and image scanning.

Supported Scanner Matrix for Supply Chain Security Tools - Scan

This topic contains limits you observe with scanners which are provided for SCST - Scan. There might be more limits which are not mentioned in the following table.

Grype

Workload Type	Impact	Potential Workarounds
.Net	**Observation:** Source Scans for .Net workloads do not show any results in the Tanzu Application Platform GUI nor the CLI. If scanning a mono repository that includes additional types of packages, such as a front-end JavaScript package, source scans might report vulnerabilities. **Reason:** Grype requires a ".deps.json" file for identifying the dependencies for scanning. Given that this file is created after the .Net project is compiled (which happens after the source scan step), doing Grype source scans on .Net workloads might not report any vulnerabilities. Review the upstream issue here.	Grype image scans for .Net workloads function in most cases. If using an out-of-the-box Supply Chain with scanning, users can select one of the following options: 1. Do nothing. Source scan might not report any vulnerabilities but image scan can. 2. Edit the Supply Chain to use an alternative scanner.
Java	**Observation:** Source Scans for Java workloads do not show any results in the Tanzu Application Platform GUI nor the CLI. **Reason:** For Java using Gradle, dependency lock files are not guaranteed, so Grype uses dependencies present in the built binaries, such as `.jar` or `.war` files. Grype fails to find vulnerabilities during a source scan because VMware discourages committing binaries to source code repositories. Review the upstream issue here.	Grype image scans for Java workloads function in most cases. If using an out-of-the-box Supply Chain with scanning, users can select one of the following options: 1. Do nothing. Source scan might not report any vulnerabilities but image scan can. 2. Edit the Supply Chain to use an alternative scanner that supports Java for source scans.

Prerequisites for Snyk Scanner for Supply Chain Security Tools - Scan (Beta)

This topic describes the prerequisites you must complete to install Supply Chain Security Tools - Scan (Snyk Scanner) from the Tanzu Application Platform package repository.

Important

Snyk's image scanning capability is in beta. Snyk might only return a partial list of CVEs when scanning Buildpack images.

Prepare the Snyk Scanner configuration

1. Obtain a Snyk API Token from the Snyk documentation.
2. Create a Snyk secret YAML file and insert the base64 encoded Snyk API token into the `snyk_token`:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: snyk-token-secret
     namespace: my-apps
   data:
     snyk_token: BASE64-SNYK-API-TOKEN
   ```

   Where `BASE64-SNYK-API-TOKEN` is the Snyk API Token obtained earlier.
3. Apply the Snyk secret YAML file by running:

   ```
   kubectl apply -f YAML-FILE
   ```

   Where `YAML-FILE` is the name of the Snyk secret YAML file you created.
4. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Snyk Scanner configuration. You can add fields as needed to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

   ```
   ---
   namespace: DEV-NAMESPACE
   targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
   snyk:
     tokenSecret:
       name: SNYK-TOKEN-SECRET
   ```

   Where:

   - `DEV-NAMESPACE` is your developer namespace.

     Note

     To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

   - `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
   - `SNYK-TOKEN-SECRET` is the name of the secret you created that contains the `snyk_token` to connect to the Snyk API. This field is required.

The Snyk Scanner integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

SCST - Store integration

Using SCST - Store Integration: To persist the results found by the Snyk Scanner, you can enable the SCST - Store integration by appending the fields to the `values.yaml` file.

The Grype and Snyk Scanner Integrations both enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether the Grype Scanner Integration is installed or not. If Tanzu Application Platform is installed using the Full Profile, the Grype Scanner Integration is installed, unless it is explicitly excluded.

- If the Grype Scanner Integration is installed in the same `dev-namespace` Snyk Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.l
ocal:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store
Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Snyk and Grype both enable store, one
must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to t
he Store Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Snyk and Grype both enable store, one
must leave importFromNamespace blank
```

- If the Grype Scanner Integration is not installed in the same `dev-namespace` Snyk Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.l
ocal:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store
Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to t
he Store Deployment.
    name: "AUTH-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

Without SCST - Store Integration: The SCST - Store integration is enabled by default. If you don't want to use this integration, explicitly deactivate the integration by appending the following field to the `values.yaml` file:

```
# ...
metadataStore:
   url: "" # Configuration is moved, so set this string to empty.
```

Sample ScanPolicy for Snyk in SPDX JSON format

1. Create a ScanPolicy YAML with a Rego file for scanner output in the SPDX JSON format. Here is a sample scan policy resource:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: snyk-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      fails := contains(notAllowedSeverities, match.relationships[_].ratedBy.ra
ting[_].severity)
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      vuln := input.vulnerabilities[_]
      ratings := vuln.relationships[_].ratedBy.rating[_].severity
      comp := vuln.relationships[_].affect.to[_]
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp, vuln.id, ratings])
    }
```

2. Apply the YAML file by running:

```
kubectl apply -n $DEV_NAMESPACE -f SCAN-POLICY-YAML
```

Note

The Snyk Scanner integration is only available for an image scan, not a source scan.

After all prerequisites are completed, follow the steps in Install another scanner for Supply Chain Security Tools - Scan to install the Snyk Scanner.

Prerequisites for Carbon Black Scanner for Supply Chain Security Tools - Scan (Beta)

This topic describes prerequisites you must complete to install Supply Chain Security Tools - Scan (Carbon Black Scanner) from the Tanzu Application Platform package repository. The Carbon Black Scanner integration is only available for an image scan, not a source scan.

Important

Carbon Black's image scanning capability is in beta. Carbon Black might only return a partial list of CVEs when scanning Buildpack images.

Prepare the Carbon Black Scanner configuration

To prepare the Carbon Black Scanner configuration before you install any scanners:

1. Obtain a Carbon Black API Token from Carbon Black Cloud.
2. Create a Carbon Black secret YAML file and insert the Carbon Black API configuration key. Obtain all values from your CBC console.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: CARBONBLACK-CONFIG-SECRET
     namespace: my-apps
   stringData:
     cbc_api_id: CBC-API-ID
     cbc_api_key: CBC-API-KEY
     cbc_org_key: CBC-ORG-KEY
     cbc_saas_url: CBC-SAAS-URL
   ```

   Where:

   - `CBC-API-ID` is the API ID obtained from CBC.
   - `CBC-API-KEY` is the API Key obtained from CBC.
   - `CBC-ORG-KEY` is the Org Key of your CBC organization.
   - `CBC-SAAS-URL` is the CBC Backend URL.
3. Apply the Carbon Black secret YAML file by running:

   ```
   kubectl apply -f YAML-FILE
   ```

   Where `YAML-FILE` is the name of the Carbon Black secret YAML file you created.
4. Define the `--values-file` flag to customize the default configuration. Create a `values.yaml` file by using the following configuration:

   You must define the following fields in the `values.yaml` file for the Carbon Black Scanner configuration. You can add fields as needed to enable or deactivate behaviors. You can append the values to this file as shown later in this topic.

   ```
   ---
   namespace: DEV-NAMESPACE
   ```

```
targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
carbonBlack:
  configSecret:
    name: CARBONBLACK-CONFIG-SECRET
```

- `DEV-NAMESPACE` is your developer namespace.

  Important

  To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `CARBONBLACK-CONFIG-SECRET` is the name of the secret you created that contains the Carbon Black configuration to connect to CBC. This field is required.

The Carbon Black Scanner integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

SCST - Store integration

To Integrate:

1. Do one of the following procedures:
   - Use the Supply Chain Security Tools - Store
   - Without using the Supply Chain Security Tools - Store
2. Apply the YAML.

Using SCST - Store Integration

To persist the results found by the Carbon Black Scanner, you can enable the SCST - Store integration by appending the fields to the `values.yaml` file.

The Grype and Carbon Black Scanner Integrations both enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether the Grype Scanner Integration is installed or not. If Tanzu Application Platform was installed using the Full Profile, the Grype Scanner Integration was installed, unless it was explicitly excluded.

- If the Grype Scanner Integration is installed in the same `dev-namespace` Carbon Black Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.l
ocal:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store
Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Carbon Black and Grype both enable st
ore, one must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
```

```
    #! The name of the secret that contains the auth token to authenticate to t
he Store Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Carbon Black and Grype both enable st
ore, one must leave importFromNamespace blank
```

- If the Grype Scanner Integration is not installed in the same `dev-namespace` Carbon Black Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.l
ocal:8443"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store
Deployment.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to t
he Store Deployment.
    name: "AUTH-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

Without SCST - Store Integration

The SCST - Store integration is enabled by default. If you don't want to use this integration, explicitly deactivate the integration by appending the following field to the `values.yaml` file:

```
# ...
metadataStore:
  url: "" # Disable Supply Chain Security Tools - Store integration
```

Sample ScanPolicy in CycloneDX format

1. Create a ScanPolicy YAML with a Rego file for scanner output in the CycloneDX format. For example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
```

```
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

2. Apply the YAML:

```
kubectl apply -n $DEV_NAMESPACE -f SCAN-POLICY-YAML
```

After you complete all prerequisites, install the Carbon Black Scanner. See Install another scanner for Supply Chain Security Tools - Scan.

Prerequisites for Prisma Scanner for Supply Chain Security Tools - Scan (Alpha)

This topic describes prerequisites you must complete to install SCST - Scan (Prisma) from the VMware package repository.

Important

This integration is in Alpha, which means that it is still in active development by the Tanzu Practices Global Tech Team and might be subject to change at any point. Users might encounter unexpected behavior.

Verify the latest alpha package version

Run this command to output a list of available tags.

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/pr
isma-repo-scanning-bundle | sort -V
```

Use the latest version returned in place of the sample version in this topic. For example, `0.1.5-alpha.13` in the following output.

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/pr
isma-repo-scanning-bundle | sort -V
0.1.4-alpha.11
0.1.4-alpha.12
0.1.4-alpha.15
0.1.5-alpha.11
0.1.5-alpha.12
0.1.5-alpha.13
```

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before installing. The Prisma Scanner is in the Alpha development phase, and not packaged as part of Tanzu Application Platform. It is hosted on the VMware Project Repository instead of VMware Tanzu Network. If you relocated the Tanzu Application Platform images, you can also relocate the Prisma Scanner package. If you don't relocate the images, the Prisma Scanner installation depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

For information about supported registries, see each registry's documentation.

To relocate images from the VMware Project Registry to your registry:

1. Set up environment variables for installation:

   ```
   export IMGPKG_REGISTRY_HOSTNAME_0=registry.tanzu.vmware.com
   export IMGPKG_REGISTRY_USERNAME_0=MY-TANZUNET-USERNAME
   export IMGPKG_REGISTRY_PASSWORD_0=MY-TANZUNET-PASSWORD
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   export VERSION=VERSION-NUMBER
   export INSTALL_REPO=TARGET-REPOSITORY
   ```

   Where:

   - `MY-REGISTRY-USER` is the user with write access to MY-REGISTRY.
   - `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
   - `MY-REGISTRY` is your own registry.
   - `VERSION` is your Prisma Scanner version. For example, `0.1.4-alpha.12`.
   - `TARGET-REPOSITORY` is your target repository, a directory or repository on `MY-REGISTRY` that serves as the location for the installation files for Prisma Scanner.

2. Install the Carvel tool imgpkg CLI. See Deploying Cluster Essentials.
3. Relocate images with the imgpkg CLI by running:

   ```
   imgpkg copy -b projects.registry.vmware.com/tanzu_practice/tap-scanners-packag
   e/prisma-repo-scanning-bundle:${VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAM
   E}/${INSTALL_REPO}/prisma-repo-scanning-bundle
   ```

Add the Prisma Scanner package repository

Tanzu CLI packages are available on repositories. Adding the Prisma Scanning package repository makes the Prisma Scanning bundle and its packages available for installation.

> **Note**
>
> VMware recommends, but does not require, relocating images to a registry for installation. This section assumes that you relocated images to a registry. See the earlier section to fill in the variables.

VMware recommends installing the Prisma Scanner objects in the existing `tap-install` namespace to keep the Prisma Scanner grouped logically with the other Tanzu Application Platform components.

1. Add the Prisma Scanner package repository to the cluster by running:

   ```
   tanzu package repository add prisma-scanner-repository \
     --url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/prisma-repo-scanning-bundl
   e:$VERSION \
     --namespace tap-install
   ```

2. Get the status of the Prisma Scanner package repository, and ensure that the status updates to `Reconcile succeeded` by running:

   ```
   tanzu package repository get prisma-scanner-repository --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package repository get prisma-scanning-repository --namespace tap-insta
   ll
   - Retrieving repository prisma-scanner-repository...
   NAME:          prisma-scanner-repository
   VERSION:       71091125
   REPOSITORY:    projects.registry.vmware.com/tanzu_practice/tap-scanners-packag
   e/prisma-repo-scanning-bundle
   TAG:           0.1.4-alpha.12
   STATUS:        Reconcile succeeded
   REASON:
   ```

3. List the available packages by running:

   ```
   tanzu package available list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list --namespace tap-install
   / Retrieving available packages...
     NAME                                                DISPLAY-NAME
   SHORT-DESCRIPTION
     prisma.scanning.apps.tanzu.vmware.com               Prisma for Supply Chain
   Security Tools - Scan                            Default scan templates using
   Prisma
   ```

Prepare the Prisma Scanner configuration

Before installing the Prisma scanner, you must create the configuration and a Kubernetes secret that contains credentials to access Prisma Cloud.

Obtain Console URL and Access Keys and Token

The Prisma Scanner supports two methods of authentication:

1) Basic Authentication with API Key and Secret 2) Token Based Authentication

The steps to configure both are outlined to allow you to decide which option you use.

> **Note**
>
> The token method issued by Prisma Cloud has a expiration of 1 hour, so it requires frequent refreshing.

To obtain your Prisma Compute Console URL and Access Keys and Token. See Access keys in the Palo Alto Networks documentation.

> **Note**
>
> Generated tokens expire after an hour.

Access key and secret authentication

To create a Prisma secret, use the following instructions.

1. Create a Prisma secret YAML file and insert the base64 encoded Prisma API token into the `prisma_token`:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: PRISMA-ACCESS-KEY-SECRET
     namespace: APP-NAME
   data:
     username: BASE64-PRISMA-ACCESS-KEY-ID
     password: BASE64-PRISMA-ACCESS-KEY-PASSWORD
   ```

   Where:

   - `PRISMA-ACCESS-KEY-SECRET` is the name of your Prisma token secret.
   - `APP-NAME` is the namespace you want to use.
   - `BASE64-PRISMA-ACCESS-KEY-ID` is your base64 encoded Prisma Access Key ID.
   - `BASE64-PRISMA-ACCESS-KEY-PASSWORD` is your base64 encoded Prisma Access Key Password.

2. Apply the Prisma secret YAML file by running:

   ```
   kubectl apply -f YAML-FILE
   ```

   Where `YAML-FILE` is the name of the Prisma secret YAML file you created.

3. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Prisma Scanner configuration. You can add fields to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

   ```
   ---
   namespace: DEV-NAMESPACE
   targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
   prisma:
     url: PRISMA-URL
   ```

```
  basicAuth:
    name: PRISMA-ACCESS-KEY-SECRET
```

Where:

- `DEV-NAMESPACE` is your developer namespace.

> **Note**
>
> To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `PRISMA-URL` is the FQDN of your Twistlock server.
- `PRISMA-CONFIG-SECRET` is the name of the secret you created that contains the Prisma configuration to connect to Prisma. This field is required.

The Prisma integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

Access Token Authentication

1. Create a Prisma secret YAML file and insert the base64 encoded Prisma API token into the `prisma_token`:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: PRISMA-TOKEN-SECRET
     namespace: APP-NAME
   data:
     prisma_token: BASE64-PRISMA-API-TOKEN
   ```

   Where:

   - `PRISMA-TOKEN-SECRET` is the name of your Prisma token secret.
   - `APP-NAME` is the namespace you want to use.
   - `BASE64-PRISMA-API-TOKEN` is the name of your base64 encoded Prisma API token.

2. Apply the Prisma secret YAML file by running:

   ```
   kubectl apply -f YAML-FILE
   ```

   Where `YAML-FILE` is the name of the Prisma secret YAML file you created.

3. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Prisma Scanner configuration. You can add fields as needed to activate or deactivate behaviors. You can append the values to this file as shown later in this topic. Create a `values.yaml` file by using the following configuration:

   ```
   ---
   namespace: DEV-NAMESPACE
   targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
   prisma:
     url: PRISMA-URL
   ```

```
  tokenSecret:
    name: PRISMA-CONFIG-SECRET
```

Where:

- `DEV-NAMESPACE` is your developer namespace.

Note

To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
- `PRISMA-URL` is the FQDN of your Twistlock server.
- `PRISMA-CONFIG-SECRET` is the name of the secret you created that contains the Prisma configuration to connect to Prisma. This field is required.

SCST - Store integration

The Prisma Scanner integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

When using SCST - Store integration, to persist the results found by the Prisma Scanner, you can enable the SCST - Store integration by appending the fields to the `values.yaml` file.

The Grype, Snyk, and Prisma Scanner Integrations enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether the Grype Scanner Integration is installed or not. If Tanzu Application Platform is installed using the Full Profile, the Grype Scanner Integration is installed unless it is explicitly excluded.

Multiple Scanners installed

In order to find your CA secret name and authentication token secret name as needed for your values.yaml when installing Prisma Scanner you must look at the configuration of a prior installed scanner in the same namespace as it already exists.

For information about how the scanner was likely initially created, see Multicluster Setup

An example `values.yaml` when there are other scanners already installed in the same `dev-namespace` where the Prisma Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:84
43"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deploym
ent.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Prisma and Grype/Snyk both enable store, one
must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Stor
```

```
e Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Prisma and Grype/Snyk both enable store, one
must leave importFromNamespace blank
```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the ca.crt to connect to the Store Deployment. Default is `app-tls-cert`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

Prisma Only Scanner Installed

For information about creating and exporting secrets for the Metadata Store CA and authentication token referenced in the data values when installing Prisma Scanner, see Multicluster Setup.

An example `values.yaml` when no other scanner integrations installed in the same `dev-namespace` where the Prisma Scanner is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:84
43"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deploym
ent.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Stor
e Deployment.
    name: "AUTH-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the `ca.crt` to connect to the Store Deployment. Default is `app-tls-cert`.
- `STORE-SECRETS-NAMESPACE` is the namespace where the secrets for the Store Deployment live. Default is `metadata-store`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

No Store Integration

If you do not want to enable the SCST - Store integration, explicitly deactivate the integration by appending the following field to the `values.yaml` file that is enabled by default:

```
# ...
metadataStore:
  url: "" # Deactivate Supply Chain Security Tools - Store integration
```

Prepare the ScanPolicy

To prepare the ScanPolicy, use the instructions in the following sections.

Sample ScanPolicy using Prisma Policies

The following sample ScanPolicy allows you to control whether the SupplyChain passes or fails based on the compliance and vulnerability rules configured in the Prisma Compute Console.

The policy reads the `complianceScanPassed` and `vulnerabilityScanPassed` fields returned from Prisma scanner output to control the results of the scan.

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: prisma-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    import future.keywords.in

    deny[msg] {
      vulnerabilityAndComplianceScanResults := {e | e := input.bom.metadata.propertie
s.property[_]}
      some result in vulnerabilityAndComplianceScanResults
      failedScans:= "false" in result
      failedScans
      vulnerabilityMessages := { message |
        components := {e | e := input.bom.components.component} | {e | e := input.bom.
components.component[_]}
        some component in components
        vulnerabilities := {e | e := component.vulnerabilities.vulnerability} | {e | e
:= component.vulnerabilities.vulnerability[_]}
        some vulnerability in vulnerabilities
        ratings := {e | e := vulnerability.ratings.rating.severity} | {e | e := vulner
ability.ratings.rating[_].severity}
        formattedRatings := concat(", ", ratings)
        message := sprintf("Vulnerability - Component: %s CVE: %s Severity: %s", [comp
onent.name, vulnerability.id, formattedRatings])
      }
      complianceMessages := { message |
        compliances := {e | e := input.bom.metadata.component.compliances.compliance}
| {e | e := input.bom.metadata.component.compliances.compliance[_]}
        some compliance in compliances
        message := sprintf("Compliance - %s \\nId: %s Severity: %s Category: %s", [com
pliance.title, compliance.id, compliance.severity, compliance.category])
      }
      combinedMessages := complianceMessages | vulnerabilityMessages
      some message in combinedMessages
      msg := message
    }
```

Sample ScanPolicy using Local Policies

The following sample ScanPolicy allows you to control whether the SupplyChain passes or fails based on the Prisma Scanner CycloneDX vulnerability results returned from the Prisma Scanner.

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: prisma-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

Apply the YAML:

```
kubectl apply -n $DEV-NAMESPACE -f SCAN-POLICY-YAML
```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `SCAN-POLICY-YAML` is the name of your SCST - Scan YAML.

Install Prisma Scanner

After all prerequisites are completed, install the Prisma Scanner. See Install another scanner for Supply Chain Security Tools - Scan.

Self-Signed Registry Certificate

When attempting to pull an image from a registry with a self-signed certificate during image scans additional configuration is necessary.

Tanzu Application Platform Values Shared CA

If your `tap-values.yaml` used during install has the following shared section filled out, Prisma Scanner uses this and enable it to connect to your registry without additional configuration.

```
shared:
   ca_cert_data: | # To be passed if using custom certificates.
      -----BEGIN CERTIFICATE-----
      MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
      -----END CERTIFICATE-----
```

Secret within Developer Namespace

1. Create a secret that holds the registry's CA certificate data.

   An example of the secret:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
    name: prisma-registry-cert
    namespace: dev
   type: Opaque
   data:
    ca_cert_data: BASE64_CERT
   ```

2. Update your Prisma Scanner install values.yaml.

   Add `caCertSecret` to the root of your `prisma-values.yaml` when installing Prisma Scanner

   Example:

   ```
   namespace: dev
   targetImagePullSecret: tap-registry
   caCertSecret: prisma-registry-cert
   ```

Connect to Prisma through a Proxy

To connect to Prisma through a proxy, you must add `environmentVariables` configuration to your `prisma-values.yaml`.

Note All valid container `env` configurations are supported.

For example:

```
namespace: dev
targetImagePullSecret: tap-registry
environmentVariables:
- name: HTTP_PROXY
  value: "test.proxy.com"
- name: HTTPS_PROXY
  value: "test.proxy.com"
- name: NO_PROXY
  value: "127.0.0.1,.svc,.svc.cluster.local,demo.app"
```

Known Limits

- OpenShift is not supported

Install Trivy for Supply Chain Security Tools - Scan (alpha)

This topic describes how you can install SCST - Scan (Trivy) from the VMware package repository.

Important

This integration is in Alpha, which means that it is still in active development by the Tanzu Practice Global Tech Team and might be subject to change at any point. Users might encounter unexpected behavior.

Verify the latest alpha package version

Run the following command to output a list of available tags.

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/tr
ivy-repo-scanning-bundle | sort -V
```

For example:

```
imgpkg tag list -i projects.registry.vmware.com/tanzu_practice/tap-scanners-package/tr
ivy-repo-scanning-bundle | sort -V

0.1.4-alpha.6
0.1.4-alpha.1
0.1.4-alpha.3
0.1.4-alpha.5
0.1.4-alpha.6
```

In this topic, use the latest version returned by the command above.

Relocate images to a registry

VMware recommends relocating the images from VMware Tanzu Network registry to your own container image registry before installing.

Trivy is in the Alpha development phase, is not packaged as part of the Tanzu Application Platform package, and is hosted on the VMware Project Repository instead of VMware Tanzu Network. If you relocated the Tanzu Application Platform images, you might also want to relocate the Trivy package.

If you don’t relocate the images, Trivy installation depends on VMware Tanzu Network for continued operation, and VMware Tanzu Network offers no uptime guarantees. The option to skip relocation is documented for evaluation and proof-of-concept only.

For information about supported registries, see the registry’s documentation.

To relocate images from the VMware Project Registry to your registry:

1. Install Docker if it is not already installed.
2. Set up environment variables for installation by running:

   ```
   export INSTALL_REGISTRY_USERNAME=MY-REGISTRY-USER
   export INSTALL_REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALL_REGISTRY_HOSTNAME=MY-REGISTRY
   ```

```
export VERSION=VERSION-NUMBER
export INSTALL_REPO=TARGET-REPOSITORY
```

Where:

- `MY-REGISTRY-USER` is the user with write access to MY-REGISTRY.
- `MY-REGISTRY-PASSWORD` is the password for `MY-REGISTRY-USER`.
- `MY-REGISTRY` is your own registry.
- `VERSION` is your Trivy version. For example, `0.1.4-alpha.6`.
- `TARGET-REPOSITORY` is your target repository, a directory or repository on `MY-REGISTRY` that serves as the location for the installation files for Trivy.

3. Install the Carvel tool imgpkg CLI. See Deploying Cluster Essentials v1.4.
4. Relocate the images with the imgpkg CLI by running:

```
imgpkg copy -b projects.registry.vmware.com/tanzu_practice/tap-scanners-packag
e/trivy-repo-scanning-bundle:${VERSION} --to-repo ${INSTALL_REGISTRY_HOSTNAME}/
${INSTALL_REPO}/trivy-repo-scanning-bundle
```

Note

The VMware project repository does not require authentication, so you don't need to perform a Docker login.

Add Trivy package repository

Tanzu CLI packages are available on repositories. Adding the Trivy scanning package repository makes the Trivy scanning bundle and its packages available for installation.

Note

VMware recommends, but does not require, relocating images to a registry for installation. The following section requires that you relocated images to a registry. See the earlier section to fill in the variables.

VMware recommends installing Trivy objects in the existing `tap-install` namespace to keep Trivy grouped logically with the other Tanzu Application Platform components.

1. Add Trivy package repository to the cluster by running:

```
tanzu package repository add trivy-scanner-repository \
--url ${INSTALL_REGISTRY_HOSTNAME}/${INSTALL_REPO}/trivy-repo-scanning-bundl
e:$VERSION \
--namespace tap-install
```

2. Get the status of Trivy package repository, and ensure that the status updates to `Reconcile succeeded` by running:

```
tanzu package repository get trivy-scanner-repository --namespace tap-install
```

For example:

```
tanzu package repository get trivy-scanner-repository --namespace tap-install
```

```
NAME:           trivy-scanner-repository
VERSION:        7750726
REPOSITORY:     projects.registry.vmware.com/tanzu_practice/tap-scanners-packag
e/trivy-repo-scanning-bundle
TAG:            0.1.4-alpha.6
STATUS:         Reconcile succeeded
REASON:
```

3. List the available packages by running:

   ```
   tanzu package available list --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list --namespace tap-install
   / Retrieving available packages...
   NAME                                                DISPLAY-NAME
   SHORT-DESCRIPTION
   trivy.scanning.apps.tanzu.vmware.com                  trivy
   Default scan templates using Trivy
   ```

Prepare Trivy configuration

Before installing the Trivy, you must create the configuration necessary to install Trivy.

1. Define the `--values-file` flag to customize the default configuration. You must define the following fields in the `values.yaml` file for the Trivy Scanner configuration. You can add fields as needed to activate or deactivate behaviors. You can append the values to the `values.yaml` file. Create a `values.yaml` file by using the following configuration:

   ```
   ---
   namespace: DEV-NAMESPACE
   targetImagePullSecret: TARGET-REGISTRY-CREDENTIALS-SECRET
   targetSourceSshSecret: TARGET-SOURCE-SSH-SECRET
   ```

   Where:

   - `DEV-NAMESPACE` is your developer namespace.

   Note

   To use a namespace other than the default namespace, ensure that the namespace exists before you install. If the namespace does not exist, the scanner installation fails.

   - `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from a private registry for scanning.
   - `TARGET-SOURCE-SSH-SECRET` is the name of the secret containing SSH credentials for cloning private repositories

2. To see all available values, run the following command using the version that you want:

   ```
   VERSION="0.1.4-alpha.6"
   tanzu package available get trivy.scanning.apps.tanzu.vmware.com/$VERSION --val
   ues-schema -n tap-install
   ```

   Example output:

```
KEY                                          DEFAULT
TYPE    DESCRIPTION
environmentVariables                         <nil>
<nil>   Environment Variables you want added to the scan container to impact tr
ivy behavior
resources.limits.cpu                         1000m
string  Limits describes the maximum amount of cpu resources allowed.
resources.requests.cpu                       250m
string  Requests describes the minimum amount of cpu resources required.
resources.requests.memory                    128Mi
string  Requests describes the minimum amount of memory resources
scanner.docker.server
string  <nil>
scanner.docker.username
string  <nil>
scanner.docker.password
string  <nil>
scanner.pullSecret
string  <nil>
scanner.serviceAccount                       trivy-scanner
string  Name of scan pod's service ServiceAccount
scanner.serviceAccountAnnotations            <nil>
<nil>   Annotations added to ServiceAccount
trivy.cli.image.additionalArguments
string  additional arguments to be appended to the image scan command
trivy.cli.plugins.aqua.repositoryUrl
string  location of the aqua plugin tar in an OCI registry to be used in place
of the embedded version
trivy.cli.repositoryUrl
string  location of the CLI tar in an OCI registry to be used in place of the e
mbedded version
trivy.cli.source.additionalArguments
string  additional arguments to be appended to the fs scan command
trivy.db.repositoryUrl
string  location of the vulnerability database in an OCI registry to be used as
the download location prior to running a scan
caCertSecret
string  Reference to the secret containing the registry ca cert set as ca_cert_
data
metadataStore.authSecret.importFromNamespace
string  Namespace from which to import the Insight Metadata Store auth_token
metadataStore.authSecret.name
string  Name of deployed Secret with key auth_token
metadataStore.caSecret.importFromNamespace   metadata-store
string  Namespace from which to import the Insight Metadata Store CA Cert
metadataStore.caSecret.name                  app-tls-cert
string  Name of deployed Secret with key ca.crt holding the CA Cert of the Insi
ght Metadata Store
metadataStore.clusterRole                    metadata-store-read-write
string  Name of the deployed ClusterRole for read/write access to the Insight M
etadata Store deployed in the same cluster
metadataStore.url                            https://metadata-store-app.metada
ta-store.svc.cluster.local:8443  string  Url of the Insight Metadata Store
namespace                                    default
string  Deployment namespace for the Scan Templates
targetImagePullSecret
string  Reference to the secret used for pulling images from private registry
targetSourceSshSecret
string  Reference to the secret containing SSH credentials for cloning private
repositories
```

SCST - Store integration

Trivy integration can work with or without the SCST - Store integration. The `values.yaml` file is slightly different for each configuration.

To persist the results found by the Trivy, enable the SCST - Store integration by appending the SCST- scan fields to Trivy`values.yaml` file.

The Grype, Snyk, Prisma, Carbon Black, and Trivy integrations enable the Metadata Store. To prevent conflicts, the configuration values are slightly different based on whether another scanner integration is installed or not. If Tanzu Application Platform is installed using the Full Profile, the Grype Scanner Integration is installed unless it is explicitly excluded.

Multiple scanners installed

When installing Trivy, find your CA secret name and authentication token secret name for your `values.yaml` ny looking at the configuration of a prior installed scanner in the same namespace as it already exists.

For information about how the scanner was initially created, see Multicluster Setup.

The following example `values.yaml` has other scanners already installed in the same `dev-namespace` where Trivy is installed:

```
#! ...
metadataStore:
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:84
43"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deploym
ent.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    importFromNamespace: "" #! since both Trivy and Grype/Snyk both enable store, one
must leave importFromNamespace blank
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Stor
e Deployment.
    name: "AUTH-SECRET-NAME"
    importFromNamespace: "" #! since both Trivy and Grype/Snyk both enable store, one
must leave importFromNamespace blank
```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the ca.crt to connect to the Store Deployment. Default is `app-tls-cert`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

Trivy is the only scanner installed

For a walk through of creating and exporting secrets for the Metadata Store CA and authentication token which referenced in the data values, see Multicluster Setup.

The following example `values.yaml` has no other scanner integrations installed in the same `dev-namespace` where Trivy is installed:

```
#! ...
metadataStore:
```

```
  #! The url where the Store deployment is accessible.
  #! Default value is: "https://metadata-store-app.metadata-store.svc.cluster.local:84
43"
  url: "STORE-URL"
  caSecret:
    #! The name of the secret that contains the ca.crt to connect to the Store Deploym
ent.
    #! Default value is: "app-tls-cert"
    name: "CA-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    #! Default value is: "metadata-store"
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
  #! authSecret is for multicluster configurations.
  authSecret:
    #! The name of the secret that contains the auth token to authenticate to the Stor
e Deployment.
    name: "AUTH-SECRET-NAME"
    #! The namespace where the secrets for the Store Deployment live.
    importFromNamespace: "STORE-SECRETS-NAMESPACE"
```

Where:

- `STORE-URL` is the URL where the Store deployment is accessible.
- `CA-SECRET-NAME` is the name of the secret that contains the `ca.crt` to connect to the Store Deployment. Default is `app-tls-cert`.
- `STORE-SECRETS-NAMESPACE` is the namespace where the secrets for the Store Deployment live. Default is `metadata-store`.
- `AUTH-SECRET-NAME` is the name of the secret that contains the authentication token to authenticate to the Store Deployment.

No store integration

If you do not want to enable the SCST - Store integration, deactivate the integration by appending the following field to the `values.yaml` file that is enabled by default:

```
# ...
metadataStore:
  url: "" # Deactivate Supply Chain Security Tools - Store integration
```

Prepare the ScanPolicy

The following sample ScanPolicy allows you to control whether the SupplyChain passes or fails based on the CycloneDX vulnerability results returned from the Trivy.

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
   name: trivy-scan-policy
   labels:
      app.kubernetes.io/part-of: enable-in-gui
spec:
   regoFile: |
      package main

      import future.keywords.in
      import future.keywords.every

      # Accepted Values: "critical", "high", "medium", "low", unknown"
      notAllowedSeverities := ["critical", "high", "unknown"]
```

```
      notAllowedSet := {x | x := notAllowedSeverities[_]}
      ignoreCves := []

      isSafe(match) {
        severities := { e | e := match.ratings.rating.severity } | { e | e := match.ra
tings.rating[_].severity }
        every severity in severities {
            not severity in notAllowedSet
        }
      }

      isIgnored(match) {
        match.id in ignoreCves
      }

      deny[msg] {
        notAllowedVulnerabilities := { vulnerability |
          vulnerabilities := {e | e := input.bom.vulnerabilities.vulnerability} | {e |
e := input.bom.vulnerabilities.vulnerability[_]}
          some vulnerability in vulnerabilities
          not isIgnored(vulnerability)
          not isSafe(vulnerability)
        }
        formattedVulnerabilityMessages := { message |
          some vulnerability in notAllowedVulnerabilities
          ratings := {e | e := vulnerability.ratings.rating.severity} | {e | e := vuln
erability.ratings.rating[_].severity}
          formattedRatings := concat(", ", ratings)
          affectedComponents := {e | e := vulnerability.affects.target.ref} | {e | e :
= vulnerability.affects.target[_].ref}
          formattedComponents := concat("\\n", affectedComponents)
          message = sprintf("CVE: %s \\nRatings: %s\\nAffected Components: \\n%s", [vu
lnerability.id, formattedRatings, formattedComponents])
        }
        some formattedVulnerabilityMessage in formattedVulnerabilityMessages
        msg := formattedVulnerabilityMessage
      }
```

To prepare the ScanPolicy:

1. Apply the following to the YAML:

   ```
   kubectl apply -n $DEV-NAMESPACE -f SCAN-POLICY-YAML
   ```

   Where:

   - `DEV-NAMESPACE` is the name of the developer namespace you want to use.
   - `SCAN-POLICY-YAML` is the name of your SCST - Scan YAML.

Install Trivy

After the following prerequisites are completed, install the Trivy:

- Prerequisites listed in Install another scanner for Supply Chain Security Tools - Scan.
- Install the ORAS CLI. See the ORAS documentation.

Air-gap configuration

This section explains how to configure Trivy in an air-gapped environment.

For information about additional flags and configuration, see Air-Gapped Environment in the Trivy documentation.

Relocate a Trivy database to your registry

Note

Using a relocated database means you are taking responsibility for keeping it up to date to ensure that security scans are relevant. Stale databases weaken your security posture.

If you have a host with access, you can use the ORAS CLI to perform a copy.

```
oras copy -r ghcr.io/aquasecurity/trivy-db:2 registry.company.com/project_name/trivy-d
b:2 # the tag of 2 is required

Copying 4a39b38cf2fd db.tar.gz
Copied  4a39b38cf2fd db.tar.gz
Copied ghcr.io/aquasecurity/trivy-db:2 => registry.company.com/project_name/trivy-db:2
Digest: sha256:ed57874a80499e858caac27fc92e4952346eb75a2774809ee989bcd2ce48897a
```

If not, you can use the ORAS CLI to download the database and manifest and then push to your registry.

1. Download the trivy-db.

   ```
   oras pull ghcr.io/aquasecurity/trivy-db:2
   ```

   Example output:

   ```
   oras pull ghcr.io/aquasecurity/trivy-db:2
   Downloading 1612cc15d377 db.tar.gz
   Downloaded  1612cc15d377 db.tar.gz
   Pulled ghcr.io/aquasecurity/trivy-db:2
   Digest: sha256:af903c7ddbe7516f18b06254b6297cf53c0ece918def07322925c71d2f694860
   ```

2. Download the manifest for trivy-db.

   ```
   oras manifest fetch ghcr.io/aquasecurity/trivy-db:2 > trivy-db-manifest.json
   ```

3. Add the media type to the manifest.

   ```
   jq '.mediaType="application/vnd.oci.image.manifest.v1+json"' trivy-db-manifest.
   json > updated-trivy-db-manifest.json
   ```

4. Push the prior downloaded trivy-db to your registry.

   ```
   oras push registry.company.com/project_name/trivy-db:2 ./db.tar.gz
   ```

   Example output:

   ```
   oras push registry.company.com/project_name/trivy-db:2 \
   ./db.tar.gz

   Uploading 1612cc15d377 db.tar.gz
   Uploaded  1612cc15d377 db.tar.gz
   Pushed registry.company.com/project_name/trivy-db:2
   Digest: sha256:41a7eeab8837e90d8a5afd56cfce73936e15d3db04c5294f992ecff9492971dc
   ```

5. Push the updated trivy-db manifest to your registry

```
oras manifest push registry.company.com/project_name/trivy-db:2 updated-trivy-d
b-manifest.json
```

Example output:

```
oras manifest push registry.company.com/project_name/trivy-db:2 updated-trivy-d
b-manifest.json
Pushed registry.company.com/project_name/trivy-db:2
Digest: sha256:b51a2fccf38e723aac1a7217ba36ca52398b2b20e3d74c9d5089dfdcd9bb2f11
```

6. Cleanup files

```
rm trivy-db-manifest.json updated-trivy-db-manifest.json db.tar.gz
```

7. Update data values with the database repository URL. Edit your `values.yaml` to add the following:

```
trivy:
db:
  repositoryUrl: "registry.company.com/project_name/trivy-db"
```

The URL leaves off the tag of `2`.

Use another Trivy version

This section describes how to use a different Trivy CLI version than what is bundled with the package.

To use another Trivy version:

1. Install the ORAS CLI. See the ORAS documentation.
2. Download the version of the CLI you are interested in from their GitHub releases page. For example: https://github.com/aquasecurity/trivy/releases/download/v0.36.0/trivy_0.36.0_Linux-64bit.tar.gz

```
wget -c https://github.com/aquasecurity/trivy/releases/download/v0.36.0/trivy_
0.36.0_Linux-64bit.tar.gz -O trivy.tar.gz
Length: 48363295 (46M) [application/octet-stream]
Saving to: 'trivy.tar.gz'

trivy.tar.gz 100%[==>]  46.12M  50.7MB/s    in 0.9s

2023-01-25 10:47:55 (50.7 MB/s) - 'trivy.tar.gz' saved [48363295/48363295]
```

3. Relocate the CLI to your registry.

   Run the following to relocate the CLI to your registry:

```
oras push registry.company.com/project_name/trivy-cli:0.36.0 \
--artifact-type trivy/cli \
./trivy.tar.gz:application/gzip

Uploading 121f4d8282aa trivy.tar.gz
Uploaded  121f4d8282aa trivy.tar.gz
Pushed registry.company.com/project_name/trivy-cli:0.36.0
Digest: sha256:5bdb18378e8f66a72f4bef4964edeccfcc2f21883e7a6caca6dbf7a3d7233696
```

4. Edit your `values.yaml` to add the location of your CLI.

```
trivy:
cli:
  repositoryUrl: "registry.company.com/project_name/trivy-cli:0.36.0"
```

Use another Trivy Aqua plug-in version

Trivy Aqua plug-in enables Aqua SaaS integration with your Trivy scans.

To use another Trivy Aqua plug-in version:

1. Install the ORAS CLI. See the ORAS documentation.
2. Download the version of Trivy Aqua plug-in you want from the GitHub releases page. See trivy-plugin-aqua in GitHub.

   For example, v0.115.14 in GitHub:

   ```
   TRIVY-AQUA-PLUGIN-VERSION="v0.115.6"
   wget -c "https://github.com/aquasecurity/trivy-plugin-aqua/releases/download/
   ${TRIVY-AQUA-PLUGIN-VERSION}/linux_amd64_${TRIVY-AQUA-PLUGIN-VERSION}.tar.gz" -
   O trivy-aqua-plugin.tar.gz

   --2023-01-30 10:44:05--  https://github.com/aquasecurity/trivy-plugin-aqua/rele
   ases/download/v0.115.6/linux_amd64_v0.115.6.tar.gz
   HTTP request sent, awaiting response... 200 OK
   Length: 50915539 (49M) [application/octet-stream]
   Saving to: ‘trivy-aqua-plugin.tar.gz’

   trivy-aqua-plugin.tar.gz 100%[==>]  48.56M  35.3MB/s    in 1.4s

   2023-01-30 10:44:07 (35.3 MB/s) - ‘trivy-aqua-plugin.tar.gz’ saved [50915539/50
   915539]
   ```

3. The YAML file is a necessary component to tell Trivy it has the plug-in already installed. Download the plugin.yml file associated with Trivy Aqua plug-in version you downloaded.

   ```
   TRIVY-AQUA-PLUGIN-VERSION="v0.115.6"
   wget -c "https://raw.githubusercontent.com/aquasecurity/trivy-plugin-aqua/${TRI
   VY-AQUA-PLUGIN-VERSION}/plugin.yaml" -O plugin.yaml

   --2023-01-30 10:46:32--  https://raw.githubusercontent.com/aquasecurity/trivy-p
   lugin-aqua/v0.115.6/plugin.yaml
   HTTP request sent, awaiting response... 200 OK
   Length: 909 [text/plain]
   Saving to: ‘plugin.yaml’
   plugin.yaml 100%[==>]     909  --.-KB/s    in 0s

   2023-01-30 10:46:32 (54.2 MB/s) - ‘plugin.yaml’ saved [909/909]
   ```

4. Relocate the plug-in and YAML to your registry:

   ```
   TRIVY-AQUA-PLUGIN-VERSION="v0.115.6"
   REPOSITORY-URL="registry.company.com/project_name/trivy-aqua-plugin:$TRIVY-AQUA
   -PLUGIN-VERSION"

   oras push ${REPOSITORY-URL} \
   --artifact-type trivy/aqua-plugin \
   ./trivy-aqua-plugin.tar.gz:application/gzip \
   ./plugin.yaml:text/yaml

   Uploading 6fb65adbfde2 plugin.yaml
   Uploading 7340855e31ff trivy-aqua-plugin.tar.gz
   Uploaded  6fb65adbfde2 plugin.yaml
   ```

```
Uploaded  7340855e31ff trivy-aqua-plugin.tar.gz
Pushed registry.company.com/project_name/trivy-aqua-plugin:v0.115.6
Digest: sha256:791274e44b97fad98edf570205fddc1b0bc21c56d3d54565ad9475fd4da969ae
```

Where:

- `TRIVY-AQUA-PLUGIN-VERSION` is the version of Trivy Aqua plug-in you are using.
- `REPOSITORY-URL` is the repository where you want to relocate the plug-in.

5. Edit your `values.yaml` to add the location of your CLI.

```
trivy:
plugins:
  aqua:
    repositoryUrl: "registry.company.com/project_name/trivy-aqua-plugin:v0.115.
6"
```

Integrate with the Aqua SaaS platform

To integrate with the Aqua SaaS platform:

1. In order to connect to the SaaS Platform you must have an API key. To create an API key:
    1. Log into Aqua SaaS.
    2. Enter CSPM.
    3. Click **Settings** -> **API Keys**.
    4. Click **Generate Key**.
    5. Save the information for the next steps.
2. To integrate with the Aqua SaaS Platform you must have an API key. You pass this to the scanner through environment variables, referenced in a secret. Create an auth secret:

    Example secret:

    ```
    apiVersion: v1
    kind: Secret
    metadata:
    name: aqua-creds
    namespace: APP-NAMESPACE
    stringData:
    aqua-key: API-KEY
    aqua-secret: API-KEY-SECRET
    ```

    Where:

    - `APP-NAMESPACE` is the developer namespace your app uses.
    - `API-KEY` is the Aqua Platform API key.
    - `API-KEY-SECRET` is the Aqua Platform API key's Secret.
3. Set environment variables to tell Trivy to connect and report to Aqua SaaS.

You can find plug-in options in the README.md in GitHub.

Here is an example of referencing your API key and secret from a Kubernetes Secret created earlier:

```
namespace: dev
targetImagePullSecret: registry-credentials
environmentVariables:
  - name: TRIVY-RUN-AS-PLUGIN
```

```
    value: aqua
  - name: AQUA-KEY
    valueFrom:
      secretKeyRef:
        name: aqua-creds
        key: aqua-key
  - name: AQUA-SECRET
    valueFrom:
      secretKeyRef:
        name: aqua-creds
        key: aqua-secret
```

Where:

- `TRIVY-RUN-AS-PLUGIN` is the Trivy plug-in you want to enable without using the subcommand.
- `AQUA-KEY` is the Aqua Platform API key.
- `AQUA-SECRET` is the Aqua Platform API key's Secret.

Self-signed registry certificate

You need additional configuration when attempting to pull an image from a registry with a self-signed certificate during image scans.

1. If your `tap-values.yaml` used during install has the following shared section filled out, Trivy uses this to connect to your registry without additional configuration. Use the following YAML with a Tanzu Application Platform values shared CA:

   ```
   shared:
   ca_cert_data: | # To be passed if using custom certificates.
       -----BEGIN CERTIFICATE-----
       MIIFXzCCA0egAwIBAgIJAJYm37SFocjlMA0GCSqGSIb3DQEBDQUAMEY...
       -----END CERTIFICATE-----
   ```

2. Create a secret that holds the registry's CA certificate data.

   An example secret:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
   name: trivy-registry-cert
   namespace: dev
   type: Opaque
   data:
   ca_cert_data: BASE64_CERT
   ```

3. Update your Trivy install `trivy-values.yaml`.

   Add `caCertSecret` to the root of your `trivy-values.yaml`.

   For example:

   ```
   namespace: dev
   targetImagePullSecret: tap-registry
   caCertSecret: trivy-registry-cert
   ```

Spec reference

This topic describes the specifications and custom resources you can use with Supply Chain Security Tools - Scan.

With the Scan Controller and Grype Scanner installed the following Custom Resource Definitions (CRDs) are now available:

```
$ kubectl get crds | grep scanning.apps.tanzu.vmware.com
imagescans.scanning.apps.tanzu.vmware.com                2021-09-09T15:22:07Z
scanpolicies.scanning.apps.tanzu.vmware.com              2021-09-09T15:22:07Z
scantemplates.scanning.apps.tanzu.vmware.com             2021-09-09T15:22:07Z
sourcescans.scanning.apps.tanzu.vmware.com               2021-09-09T15:22:07Z
```

For more information about installing SCST - Scan, see Installing Individual Packages.

About source and image scans

Both SourceScan (`sourcescans.scanning.apps.tanzu.vmware.com`) and ImageScan (`imagescans.scanning.apps.tanzu.vmware.com`) define what will be scanned, and ScanTemplate (`scantemplates.scanning.apps.tanzu.vmware.com`) will define how to run a scan. We have provided five custom resources (CRs) pre-installed for use. You can either use them as-is or as samples to create your own.

To view the pre-installed Scan Template CRs, run:

```
kubectl get scantemplates
```

You will see the following scan templates:

CR Name	Use Case
`public-source-scan-template`	Clones and scans source code from a public repository.
`private-source-scan-template`	Connects with SSH credentials to clone and scan source code from a private repository.
`public-image-scan-template`	Pulls and scans images from a public registry.
`private-image-scan-template`	Connects with the registry credentials to pull and scan images from a private registry.
`blob-source-scan-template`	To be used in a Supply Chain. Gets a `.tar.gz` available file with `wget`, uncompresses it, and scans the source code inside it.

By default, three scan templates are deployed (`public-source-scan-template`, `public-image-scan-template`, and `blob-source-scan-template`).

If `targetImagePullSecret` is set in `tap-values.yaml`, `private-image-scan-template` is also deployed. If `targetSourceSshSecret` is set in `tap-values.yaml`, `private-source-scan-template` is also deployed.

The private scan templates reference secrets created using the Docker server and credentials you provided, which means they are ready to use immediately.

For more information about the `SourceScan` and `ImageScan` CRDs and how to customize your own, refer to Configuring Code Repositories and Image Artifacts to be Scanned.

About policy enforcement around vulnerabilities found

The Scan Controller supports policy enforcement by using an Open Policy Agent (OPA) engine. ScanPolicy (`scanpolicies.scanning.apps.tanzu.vmware.com`) allows scan results to be validated for

company policy compliance and can prevent source code from being built or images from being deployed.

For more information, see Configuring Policy Enforcement using Open Policy Agent (OPA).

Scan samples for Supply Chain Security Tools - Scan

This section provides samples on multiple use cases for SCST - Scan that you can copy to your cluster for testing purposes.

- Running a sample public image scan with compliance check
- Running a sample public source scan with compliance check
- Running a sample private image scan
- Running a sample private source scan
- Running a sample public source scan of a blob/tar file

Scan samples for Supply Chain Security Tools - Scan

This section provides samples on multiple use cases for SCST - Scan that you can copy to your cluster for testing purposes.

- Running a sample public image scan with compliance check
- Running a sample public source scan with compliance check
- Running a sample private image scan
- Running a sample private source scan
- Running a sample public source scan of a blob/tar file

Sample public image scan with compliance check for Supply Chain Security Tools - Scan

This topic includes an example public image scan with compliance check for SCST - Scan.

Public image scan

The following example performs an image scan on an image in a public registry. This image revision has 223 known vulnerabilities (CVEs), spanning a number of severities. ImageScan uses the ScanPolicy to run a compliance check against the CVEs.

The policy in this example is set to only consider `Critical` severity CVEs as a violation, which returns 21 Critical Severity Vulnerabilities.

Note

This example ScanPolicy is deliberately constructed to showcase the features available and must not be considered an acceptable base policy.

In this example, the scan does the following:

- Finds all 223 of the CVEs
- Ignores any CVEs with severities that are not critical
- Indicates in the `Status.Conditions` that 21 CVEs have violated policy compliance

Define the ScanPolicy and ImageScan

Create `sample-public-image-scan-with-compliance-check.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: sample-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-public-image-scan-with-compliance-check
spec:
  registry:
    image: "nginx:1.16"
  scanTemplate: public-image-scan-template
  scanPolicy: sample-scan-policy
```

(Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For more information about setting up a watch, see Observing and Troubleshooting.

Deploy the resources

```
kubectl apply -f sample-public-image-scan-with-compliance-check.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View the scan results

```
kubectl describe imagescan sample-public-image-scan-with-compliance-check -n DEV-NAMES
PACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Note

The `Status.Conditions` includes a `Reason: EvaluationFailed` and `Message: Policy violated because of 21 CVEs`.

For more information about scan status conditions, see Viewing and Understanding Scan Status Conditions.

Edit the ScanPolicy

To edit the Scan Policy, see Step 5: Sample Public Source Code Scan with Compliance Check.

Clean up

To clean up, run:

```
kubectl delete -f sample-public-image-scan-with-compliance-check.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Sample public source code scan with compliance check for Supply Chain Security Tools - Scan

This topic includes an example public source code scan with compliance check for SCST - Scan.

Public source scan

This example performs a source scan on a public repository. The source revision has 192 known Common Vulnerabilities and Exposures (CVEs), spanning several severities. SourceScan uses the ScanPolicy to run a compliance check against the CVEs.

The example policy is set to only consider `Critical` severity CVEs as violations, which returns 7 Critical Severity Vulnerabilities.

Note

This example ScanPolicy is deliberately constructed to showcase the features available and must not be considered an acceptable base policy.

For this example, the scan (at the time of writing):

- Finds all 192 of the CVEs.
- Ignores any CVEs that have severities that are not critical.
- Indicates in the `Status.Conditions` that 7 CVEs have violated policy compliance.

Run an example public source scan

To perform an example source scan on a public repository:

1. Create `sample-public-source-scan-with-compliance-check.yaml` to define the ScanPolicy and SourceScan:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: sample-scan-policy
  labels:
    'app.kubernetes.io/part-of': 'enable-in-gui'
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Critical"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
```

```
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }

---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: sample-public-source-scan-with-compliance-check
spec:
  git:
    url: "https://github.com/houndci/hound.git"
    revision: "5805c650"
  scanTemplate: public-source-scan-template
  scanPolicy: sample-scan-policy
```

2. (Optional) Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

   ```
   watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolici
   es -n DEV-NAMESPACE
   ```

   Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

   For more information, see Observing and Troubleshooting.

3. Deploy the resources by running:

   ```
   kubectl apply -f sample-public-source-scan-with-compliance-check.yaml -n DEV-NA
   MESPACE
   ```

   Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

4. When the scan completes, view the results by running:

   ```
   kubectl describe sourcescan sample-public-source-scan-with-compliance-check -n
   DEV-NAMESPACE
   ```

   The `Status.Conditions` includes a `Reason: EvaluationFailed` and `Message: Policy violated because of 7 CVEs`. For more information, see Viewing and Understanding Scan Status Conditions.

5. If the failing CVEs are acceptable or the build must be deployed regardless of these CVEs, the app is patched to remove the vulnerabilities. Update the `ignoreCVEs` array in the ScanPolicy to include the CVEs to ignore:

   ```
   ...
   spec:
     regoFile: |
       package policies

       default isCompliant = false

       # Accepted Values: "UnknownSeverity", "Critical", "High", "Medium", "Low",
   "Negligible"
       violatingSeverities := ["Critical"]
       # Adding the failing CVEs to the ignore array
       ignoreCVEs := ["CVE-2018-14643", "GHSA-f2jv-r9rf-7988", "GHSA-w457-6q6x-cgp
   9", "CVE-2021-23369", "CVE-2021-23383", "CVE-2020-15256", "CVE-2021-29940"]
   ...
   ```

6. The changes applied to the new ScanPolicy trigger the scan to run again. Reapply the resources by running:

```
kubectl apply -f sample-public-source-scan-with-compliance-check.yaml -n DEV-NA
MESPACE
```

7. Re-describe the SourceScan CR by running:

```
kubectl describe sourcescan sample-public-source-scan-with-compliance-check -n
DEV-NAMESPACE
```

8. Ensure that `Status.Conditions` now includes a `Reason: EvaluationPassed` and `No CVEs were found that violated the policy`. You can update the `violatingSeverities` array in the ScanPolicy if you want. For reference, the Grype scan returns the following Severity spread of vulnerabilities:
    - Critical: 7
    - High: 88
    - Medium: 92
    - Low: 5
    - Negligible: 0
    - UnknownSeverity: 0
9. Clean up by running:

```
kubectl delete -f sample-public-source-scan-with-compliance-check.yaml -n DEV-N
AMESPACE
```

Sample private image scan for Supply Chain Security Tools - Scan

This example describes how you can perform a scan against an image located in a private registry for SCST - Scan.

Define the resources

Set up target image pull secret

1. Confirm that target image secret is configured. This is completed during Tanzu Application Platform installation. If the target image secret exists, see Create the private image scan.
2. If the target image secret was not configured, create a secret containing the credentials used to pull the target image you want to scan. For information about secret creation, see the Kubernetes documentation.

```
kubectl create secret docker-registry TARGET-REGISTRY-CREDENTIALS-SECRET \
--docker-server=YOUR-REGISTRY-SERVER \
--docker-username=YOUR-NAME \
--docker-password=YOUR-PASSWORD \
--docker-email=YOUR-EMAIL \
-n DEV-NAMESPACE
```

Where:

- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that is created.
- `DEV-NAMESPACE` is the developer namespace where the scanner is installed.
- `YOUR-REGISTRY-SERVER` is the registry server you want to use.

- `YOUR-NAME` is the name associated with the secret.
- `YOUR-PASSWORD` is the password associated with the secret.
- `YOUR-EMAIL` is the email associated with the secret.

3. Update the `tap-values.yaml` file to include the name of secret created earlier.

```
grype:
namespace: "MY-DEV-NAMESPACE"
targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
```

4. Upgrade Tanzu Application Platform with the modified `tap-values.yaml` file.

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP-VERSION}  -
-values-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the Tanzu Application Platform version.

Create the private image scan

Create `sample-private-image-scan.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ImageScan
metadata:
  name: sample-private-image-scan
spec:
  registry:
    image: IMAGE-URL
  scanTemplate: private-image-scan-template
```

Where `IMAGE-URL` is the URL of an image in a private registry.

(Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For more information, see Observing and Troubleshooting.

Deploy the resources

```
kubectl apply -f sample-private-image-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View the scan results

When the scan completes, run:

```
kubectl describe imagescan sample-private-image-scan -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Note

The `Status.Conditions` includes a `Reason: JobFinished` and `Message: The scan job finished`. See Viewing and Understanding Scan Status Conditions.

Clean up

```
kubectl delete -f sample-private-image-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View vulnerability reports

After completing the scans, query the Supply Chain Security Tools - Store to view your vulnerability results.

Sample private source scan for Supply Chain Security Tools - Scan

This example shows how you can perform a private source scan for SCST - Scan.

Define the resources

1. Create a Kubernetes secret with an SSH key for cloning a Git repository. See the Kubernetes documentation.

   ```
   cat <<EOF | kubectl create -f -
   apiVersion: v1
   kind: Secret
   metadata:
   name: SECRET-SSH-AUTH
   namespace: DEV-NAMESPACE
   annotations:
     tekton.dev/git-0: https://github.com
     tekton.dev/git-1: https://gitlab.com
   type: kubernetes.io/ssh-auth
   stringData:
   ssh-privatekey: |
     -----BEGIN OPENSSH PRIVATE KEY-----
     ....
     ....
     -----END OPENSSH PRIVATE KEY-----
   EOF
   ```

   Where:

   - `SECRET-SSH-AUTH` is the name of the secret that is being created.
   - `DEV-NAMESPACE` is the developer namespace where the scanner is installed.
   - `.stringData.ssh-privatekey` contains the private key with pull-permissions.

2. Update the `tap-values.yaml` file to include the name of secret created above.

   ```
   grype:
   namespace: "MY-DEV-NAMESPACE"
   targetSourceSshSecret: "SECRET-SSH-AUTH"
   ```

3. Upgrade Tanzu Application Platform with the modified `tap-values.yaml` file.

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v ${TAP-VERSION}  -
-values-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the Tanzu Application Platform version.

4. Create `sample-private-source-scan.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
name: sample-private-source-scan
spec:
git:
  url: URL
  revision: REVISION
  knownHosts: |
    KNOWN-HOSTS
scanTemplate: private-source-scan-template
```

Where:

- `URL` is the Git clone repository using SSH.
- `REVISION` is the commit hash.
- `KNOWN-HOSTS` are the SSH client stored host keys generated by ssh-keyscan.
    - For example, `ssh-keyscan github.com` produces:

```
github.com ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAq2A7hRGmdnm9tUDbO9I
DSwBK6TbQa+PXYPCPy6rbTrTtw7PHkccKrpp0yVhp5HdEIcKr6pLlVDBfOLX9QUsyC
OV0wzfjIJNlGEYsdlLJizHhbn2mUjvSAHQqZETYP81eFzLQNnPHt4EVVUh7VfDESU8
4KezmD5QlWpXLmvU31/yMf+Se8xhHTvKSCZIFImWwoG6mbUoWf9nzpIoaSjB+weqqU
UmpaaasXVal72J+UX2B+2RPW3RcT0eOzQgqlJL3RKrTJvdsjE3JEAvGq3lGHSZXy28
G3skua2SmVi/w4yCE6gbODqnTWlg7+wC604ydGXA8VJiS5ap43JXiUFFAaQ==
github.com ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAA
IbmlzdHAyNTYAAABBBEmKSENjQEezOmxkZMy7opKgwFB9nkt5YRrYMjNuG5N87uRgg
6CLrbo5wAdT/y6v0mKV0U2w0WZ2YB/++Tpockg=
github.com ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIOMqqnkVzrm0SdG6UOo
qKLsabgH5C9okWi0dh2l9GKJl
```

For example:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
name: sample-private-source-scan
spec:
git:
  url: git@github.com:acme/website.git
  revision: 25as5e7df56c6401111be514a2f3666179ba04d0
  knownHosts: |
    10.254.171.53 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItb
POVVQF/CzuAeQNv4fZVf2pLxpGHle15zkpxOosckequUDxoq
scanTemplate: private-source-scan-template
```

(Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

See Observing and Troubleshooting.

Deploy the resources

```
kubectl apply -f sample-private-source-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View the scan status

After the scan has completed, run:

```
kubectl describe sourcescan sample-private-source-scan -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Notice the `Status.Conditions` includes a `Reason: JobFinished` and `Message: The scan job finished`. See Viewing and Understanding Scan Status Conditions.

Clean up

```
kubectl delete -f sample-private-source-scan.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View vulnerability reports

After completing the scans, query the Supply Chain Security Tools - Store to view your vulnerability results.

Sample public source scan of a blob for Supply Chain Security Tools - Scan

You can do a public source scan of a blob for SCST - Scan. This example performs a scan against source code in a `.tar.gz` file. This is helpful in a Supply Chain, where there is a `GitRepository` step that handles cloning a repository and outputting the source code as a compressed archive.

Define the resources

Create `public-blob-source-example.yaml`:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  name: public-blob-source-example
spec:
  blob:
```

```
    url: "https://gitlab.com/nina-data/ckan/-/archive/master/ckan-master.tar.gz"
  scanTemplate: blob-source-scan-template
```

(Optional) Set up a watch

Before deploying the resources to a user specified namespace, set up a watch in another terminal to view the progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

For more information, see Observing and Troubleshooting.

Deploy the resources

```
kubectl apply -f public-blob-source-example.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View the scan results

When the scan completes, perform:

```
kubectl describe sourcescan public-blob-source-example -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Notice the `Status.Conditions` includes a `Reason: JobFinished` and `Message: The scan job finished`.

For more information, see Viewing and Understanding Scan Status Conditions.

Clean up

```
kubectl delete -f public-blob-source-example.yaml -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

View vulnerability reports

After completing the scans, query the Supply Chain Security Tools - Store to view your vulnerability results.

Using Grype in air-gapped (offline) environments for Supply Chain Security Tools - Scan

This topic tells you how to use Grype in air-gapped (offline) environments for Supply Chain Security Tools (SCST) - Scan.

The `grype` CLI attempts to perform two over the Internet calls:

- One to verify for later versions of the CLI.
- One to update the vulnerability database before scanning.

For the `grype` CLI to function in an offline or air-gapped environment, the vulnerability database must be hosted within the environment. You must configure the `grype` CLI with the internal URL.

The `grype` CLI accepts environment variables to satisfy these needs.

Host the Grype vulnerability database

To host Grype's vulnerability database in an air-gapped environment:

1. Retrieve Grype's listing file from its public endpoint: https://toolbox-data.anchore.io/grype/databases/listing.json.
2. Create your own `listing.json` file.

   Note Different Grype versions require specific database schema versions. To avoid compatibility issues between different versions, include a database schema for each version. For example:


```
    {
      "available": {
        "1": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 1,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v1_2023-06-16T01:33:30Z_1621f4169ffd15bea9e5.tar.gz",
            "checksum": "sha256:3f2c1b432945cca9a69b2e604f6fb231fec450fdd27f494
6fc5608692b63a9d1"
          }
        ],
        "2": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 2,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v2_2023-06-16T01:33:30Z_d6eee5e78d9b78285e1a.tar.gz",
            "checksum": "sha256:7b7e3a2a7712c72b8c5cc777733c4d8d140d8cfee65e4f0
4540abbdfe3ef1f65"
          }
        ],
        "3": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 3,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v3_2023-06-16T01:33:30Z_f96ae38a7b05987c3ece.tar.gz",
            "checksum": "sha256:8ea9fae3fda3bf3bf35bd5e5eb656fc127b59cd3c42db4c
36795556aab8a9cf0"
          }
        ],
        "4": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 4,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
ty-db_v4_2023-06-16T01:33:30Z_13bba2fa8ff62b7f8b26.tar.gz",
            "checksum": "sha256:3b53d20241b88e5aa45feb817b325c53d6efbe9fa1fc5a6
7eeddaecafa7687e0"
          }
        ],
        "5": [
          {
            "built": "2023-06-16T01:33:30Z",
            "version": 5,
            "url": "https://toolbox-data.anchore.io/grype/databases/vulnerabili
```

```
ty-db_v5_2023-06-16T01:33:30Z_e07da3853f6db6eb1104.tar.gz",
            "checksum": "sha256:93d4d9d2f9e39f86570f832cf85b7149a949ca6f1613581
b10c12393509d884f"
          }
        ]
      }
    }
```

Where `url` points to a tarball containing Grype's `vulnerability`, `db`, and `metadata.json` files.

3. Download and host the tarballs in your internal file server.

Note

Some storage solutions for internal file servers change the name of TAR files automatically because of their limits. Notice these modified names and reflect the changes in the `url`. Ensure that the timestamp in the name is correctly formatted because Grype parses the name of TAR artifact to get the timestamp.

4. Update the download `url` to point at your internal endpoint.

   For information about setting up an offline vulnerability database, see the Anchore Grype README in GitHub.

To enable Grype in offline air-gapped environments

1. Add the following to your `tap-values.yaml` file:

```
grype:
  db:
    dbUpdateUrl: INTERNAL-VULN-DB-URL
```

   Where `INTERNAL-VULN-DB-URL` is the URL that points to the internal file server.

2. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Configure Grype environmental variables

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-environmental-variables
  namespace: tap-install
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
```

```
="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

Where `spec.template.initContainers[]` specifies setting one or more environment variables in the `scan-plugin` initContainer.

Note

If you are using the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, you must import the overlay `Secret`. See Import overlay secrets.

Troubleshooting

ERROR failed to fetch latest cli version

Note

This message is a warning and the Grype scan still runs with this message.

The Grype CLI checks for later versions of the CLI by contacting the anchore endpoint over the Internet.

```
ERROR failed to fetch latest version: Get "https://toolbox-data.anchore.io/grype/relea
ses/latest/VERSION": dial tcp: lookup toolbox-data.anchore.io on [::1]:53: read udp
[::1]:65010->[::1]:53: read: connection refused
```

Solution

To deactivate this check, set the environment variable `GRYPE_CHECK_FOR_APP_UPDATE` to `false` by using a package overlay with the following steps:

1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: grype-airgap-deactivate-cli-check-overlay
     namespace: tap-install #! namespace where tap is installed
   stringData:
     patch.yaml: |
       #@ load("@ytt:overlay", "overlay")

       #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1+"
       ---
       spec:
         template:
           initContainers:
             #@overlay/match by=overlay.subset({"name": "scan-plugin"}), expects
   ="1+"
             - name: scan-plugin
   ```

```
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
              - name: GRYPE_CHECK_FOR_APP_UPDATE
                value: "false"
```

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```
package_overlays:
  - name: "grype"
    secrets:
        - name: "grype-airgap-deactivate-cli-check-overlay"
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Database is too old

```
1 error occurred:
  * db could not be loaded: the vulnerability database was built N days/weeks ago (max
allowed age is 5 days)
```

Grype needs up-to-date vulnerability information to provide accurate matches. By default, it fails to run if the local database was not built in the last 5 days.

Solution

Two options to resolve this:

1. Stale databases weaken your security posture. VMware recommends updating the database daily as the first recommended solution.
2. If updating the database daily is not an option, the data staleness check is configurable by using the environment variable `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` and is addressed using a package overlay with the following steps:
    1. Create a secret that contains the ytt overlay to add the Grype environment variable to the ScanTemplates.

```
apiVersion: v1
kind: Secret
metadata:
  name: grype-airgap-override-stale-db-overlay
  namespace: tap-install #! namespace where tap is installed
stringData:
  patch.yaml: |
    #@ load("@ytt:overlay", "overlay")

    #@overlay/match by=overlay.subset({"kind":"ScanTemplate"}),expects="1
+"
    ---
    spec:
      template:
        initContainers:
          #@overlay/match by=overlay.subset({"name": "scan-plugin"}), exp
ects="1+"
          - name: scan-plugin
            #@overlay/match missing_ok=True
            env:
              #@overlay/append
```

```
                - name: GRYPE_DB_MAX_ALLOWED_BUILT_AGE #! see note on best
practices
                  value: "120h"
```

Note

The default maximum allowed built age of Grype's vulnerability database is 5 days. This means that scanning with a 6 day old database causes the scan to fail. You can use the `GRYPE_DB_MAX_ALLOWED_BUILT_AGE` parameter to override the default in accordance with your security posture.

2. Configure `tap-values.yaml` to use `package_overlays`. Add the following to your `tap-values.yaml` file:

```
package_overlays:
  - name: "grype"
    secrets:
        - name: "grype-airgap-override-stale-db-overlay"
```

3. Update Tanzu Application Platform:

```
tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Vulnerability database is invalid

```
scan-pod[scan-plugin]  1 error occurred:
scan-pod[scan-plugin]  * failed to load vulnerability db: vulnerability database is in
valid (run db update to correct): database metadata not found: /.cache/grype/db/5
```

Solution

Examine the `listing.json` file you created. This matches the format of the listing file. The listing file is located at Anchore Grype's public endpoint. See the Grype README.md in GitHub.

An example `listing.json`:

```
{
  "available": {
    "5": [
      {
        "built": "2023-03-28T01:29:38Z",
        "version": 5,
        "url": "https://toolbox-data.anchore.io/grype/databases/vulnerability-db_v5_20
23-03-28T01:29:38Z_e49d318c32a6113eed07.tar.gz",
        "checksum": "sha256:408ce2932f04dee929a5df524e92494f2d635c6b19e30ff9f0a50425b1
fc29a1"
      },
      .....
    ]
  }
}
```

Where:

- `5` refers to the Grype's vulnerability database schema.
- `built` is the build timestamp in the format `yyyy-MM-ddTHH:mm:ssZ`.

- `url` is the download URL for the tarball containing the database. This points at your internal endpoint. The tarball contains the following files:
    - `vulnerability.db` is an SQLite file that is Grype's vulnerability database. Each time the data shape of the vulnerability database changes, a new schema is created. Different Grype versions require specific database schema versions. For example, Grype `v0.54.0` requires database schema version v5.
    - `metadata.json` file
- `checksum` is the SHA used to verify the database's integrity.

Verify these possible reasons why the vulnerability database is not valid:

1. The database schema is invalid. Confirm that the required database schema for the installed Grype version is used. Confirm that the top level version key matches the nested `version`. For example, the top level version `1` in the following snippet does not match the nested `version: 5`.

    ```
    {
      "available": {
        "1": [{
                "built": "2023-02-08T08_17_20Z",
                "version": 5,
                "url": "https://INTERNAL-ENDPOINT/PATH-TO-TARBALL/vulnerability-db_v
    5_2023-02-08T08_17_20Z_6ef73016d160043c630f.tar.gz",
                "checksum": "sha256:aab8d369933c845878ef1b53bb5c26ee49b91ddc5cd87c9e
    b57ffb203a88a72f"
        }]
      }
    }
    ```

    Where `PATH-TO-TARBALL` is the path to the tarball containing the vulnerability database.

    As stale databases weaken your security posture, VMware recommends using the newest entry of the relevant schema version in the `listing.json` file. See Anchore's grype-db in GitHub.

2. The `built` parameters in the `listing.json` file are incorrectly formatted. The proper format is `yyyy-MM-ddTHH:mm:ssZ`.
3. The `url` that you modified to point at an internal endpoint is not reachable from within the cluster. For information about verifying connectivity, see Debug Grype database in a cluster.
4. Verify if there are syntax errors in the listing.json:

    ```
    grype db check
    ```

5. Validate the configured listing.json:

    ```
    grype db list -o raw
    ```

Debug Grype database in a cluster

1. Describe the failed source scan or image scan to verify the name of the ScanTemplate being used.
    - For `sourcescan`, run:

        ```
        kubectl describe sourcescan SCAN-NAME -n DEV-NAMESPACE
        ```

- For `imagescan`, run:

  ```
  kubectl describe imagescan SCAN-NAME -n DEV-NAMESPACE
  ```

Where `SCAN-NAME` is the name of the source or image scan that failed.

2. Pause reconciliation of the `grype.scanning.apps.tanzu.vmware.com` package:

   ```
   kctrl package installed pause -i <PACKAGE-INSTALL-NAME> -n tap-install
   ```

   Where `PACKAGE-INSTALL-NAME` is the name of the `grype.scanning.apps.tanzu.vmware.com` package (e.g. grype)

3. Edit the ScanTemplate's `scan-plugin` container to include a "sleep" entrypoint which allows you to troubleshoot inside the container:

   ```
   - name: scan-plugin
     volumeMounts:
       ...
     image: #@ data.values.scanner.image
     imagePullPolicy: IfNotPresent
     env:
       ...
     command: ["/bin/bash"]
     args:
     - "sleep 1800" # insert 30 min sleep here
   ```

4. Re-run the scan.

5. Get the name of the `scan-plugin` pod.

   ```
   kubectl get pods -n DEV-NAMESPACE
   ```

6. Get a shell to the container.

   ```
   kubectl exec --stdin --tty SCAN-PLUGIN-POD -c step-scan-plugin -- /bin/bash
   ```

   Where `SCAN-PLUGIN-POD` is the name of the `scan-plugin` pod. For more information, see the Kubernetes documentation.

7. Inside the container, run Grype CLI commands to report database status and verify connectivity from the cluster to the mirror. See the Grype documentation in GitHub.

   - Report current status of Grype's database, such as location, build date, and checksum:

     ```
     grype db status
     ```

8. Ensure that the built parameters in the `listing.json` has timestamps in this proper format `yyyy-MM-ddTHH:mm:ssZ`.

9. After you complete troubleshooting, use the following command to trigger reconciliation:

   ```
   kctrl package installed kick -i <PACKAGE-INSTALL-NAME> -n tap-install
   ```

   Where `PACKAGE-INSTALL-NAME` is the name of the `grype.scanning.apps.tanzu.vmware.com` package, such as Grype.

Grype package overlays are not applied to scantemplates created by Namespace Provisioner

If you used the Namespace Provisioner to provision a new developer namespace and want to apply a package overlay for Grype, see Import overlay secrets.

Triage and Remediate CVEs for Supply Chain Security Tools - Scan

This topic explains how you can triage and remediate CVEs related to SCST - Scan.

Confirm that Supply Chain stopped due to failed policy enforcement

To confirm that Supply Chain failure is related to policy enforcement:

1. Verify that the status of the workload is `MissingValueAtPath` due to waiting on a `.status.compliantArtifact` from either the SourceScan or ImageScan:

   ```
   kubectl describe workload WORKLOAD-NAME -n DEVELOPER-NAMESPACE
   ```

2. Describe the SourceScan or ImageScan to determine what CVE(s) violated the ScanPolicy:

   ```
   kubectl describe sourcescan NAME -n DEVELOPER-NAMESPACE
   kubectl describe imagescan NAME -n DEVELOPER-NAMESPACE
   ```

Triage

The goal of triage is to analyze and prioritize the reported vulnerability data to discover the appropriate course of action to take at the remediation step. To remediate efficiently and appropriately, you need context on the vulnerabilities that are blocking your supply chain, the packages that are affected, and the impact they can have.

During triage, review which packages are impacted by the CVEs that violated your scan policy. Enabling CVE scan causes Supply Chain Choreographer by using Tanzu Application Platform GUI to visualize your supply chain, including the scans, scan policy, and CVEs. See Enable CVE scan results. You can also use the Tanzu Insight plug-in to query packages and CVEs using a CLI. See Tanzu Insight plug-in.

During this stage, VMware recommends reviewing information pertaining to the CVEs from sources such as the National Vulnerability Database or the release page of a package.

Remediation

After triage is complete, the next step is to remediate the blocking vulnerabilities quickly. Some common methods for CVE remediation are as follows:

- Updating the affected component to remove the CVE
- Amending the scan policy with an exception if you decide to accept the CVE and unblock your supply chain

Updating the affected component

Vulnerabilities that occur in older versions of a package might be resolved in later versions. Apply a patch by upgrading to a later version. You can further adopt security best practices by using your project's package manager tools, such as `go mod graph` for projects in Go, to identify transitive or indirect dependencies that can affect CVEs.

Amending the scan policy

If you decide to proceed without remediating the CVE, for example, when a CVE is evaluated to be a false positive or when a fix is not available, you can amend the ScanPolicy to ignore one or more CVEs. For information about common scanner limitations, see Note on Vulnerability Scanners. For information about templates, see Writing Policy Templates.

Under RBAC, users with the `app-operator-scanning` role that is part of the `app-operator` aggregate role, have permission to edit the ScanPolicy. See Detailed role permissions breakdown.

Observe Supply Chain Security Tools - Scan

This topic outlines observability and troubleshooting methods and issues you can use with SCST - Scan components.

Observability

The scans run inside a Tekton TaskRun where the TaskRun creates a pod. Both the TaskRun and pod are cleaned up after completion.

Before applying a new scan, you can set a watch on the TaskRuns, Pods, SourceScans, and Imagescans to observe their progression:

```
watch kubectl get sourcescans,imagescans,pods,taskruns,scantemplates,scanpolicies -n D
EV-NAMESPACE
```

Where `DEV-NAMESPACE` is the developer namespace where the scanner is installed.

Troubleshoot Supply Chain Security Tools - Scan

This topic describes troubleshooting methods you can use with SCST - Scan.

Debugging commands

Run these commands to get more logs and details about the errors around scanning. The TaskRuns and pods persist for a predefined amount of seconds before getting deleted.
(`deleteScanJobsSecondsAfterFinished` is the tap pkg variable that defines this)

Debugging Tekton TaskRun

To retrieve TaskRun events:

```
kubectl describe taskrun TASKRUN-NAME -n DEV-NAMESPACE
```

Where:

- `TASKRUN-NAME` is the name of the TaskRun.
- `DEV-NAMESPACE` is the name of the developer namespace you want to use.

Debugging Scan pods

Run the following to get error logs from a pod when scan pods are in a failing state:

```
kubectl logs scan-pod-name -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

See here for more details about debugging Kubernetes pods.

The following is an example of a successful scan run output:

```
scan:
  cveCount:
    critical: 20
    high: 120
    medium: 114
    low: 9
    unknown: 0
  scanner:
    name: Grype
    vendor: Anchore
    version: v0.37.0
  reports:
  - /workspace/scan.xml
eval:
  violations:
  - CVE node-fetch GHSA-w7rc-rwvf-8q5r Low
store:
  locations:
  - https://metadata-store-app.metadata-store.svc.cluster.local:8443/api/sources?repo=
hound&sha=5805c6502976c10f5529e7f7aeb0af0c370c0354&org=houndci
```

A scan run that has an error means that one of the init containers: `scan-plugin`, `metadata-store-plugin`, `compliance-plugin`, `summary`, or any other additional containers had a failure.

To inspect for a specific init container in a pod:

```
kubectl logs scan-pod-name -n DEV-NAMESPACE -c init-container-name
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

See Debug Init Containers in the Kubernetes documentation for debug init container tips.

Debugging SourceScan and ImageScan

To retrieve status conditions of an SourceScan and ImageScan, run:

```
kubectl describe sourcescan SOURCE-SCAN -n DEV-NAMESPACE
```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `SOURCE-SCAN` is the name of the SourceScan you want to use.

```
kubectl describe imagescan IMAGE-SCAN -n DEV-NAMESPACE
```

Where:

- `DEV-NAMESPACE` is the name of the developer namespace you want to use.
- `IMAGE-SCAN` is the name of the ImageScan you want to use.

Under `Status.Conditions`, for a condition look at the “Reason”, “Type”, “Message” values that use the keyword “Error” to investigate issues.

Debugging Scanning within a SupplyChain

See here for Tanzu workload commands for tailing build and runtime logs and getting workload status and details.

Viewing the Scan-Controller manager logs

To retrieve scan-controller manager logs:

```
kubectl -n scan-link-system logs -f deployment/scan-link-controller-manager -c manager
```

Restarting Deployment

If you encounter an issue with the scan-link controller not starting, run the following to restart the deployment to see if it's reproducible or flaking upon starting:

```
kubectl rollout restart deployment scan-link-controller-manager -n scan-link-system
```

Troubleshooting scanner to MetadataStore configuration

Insight CLI failed to post scan results to metadata store due to failed certificate verification

If you encounter this issue:

```
✖  Error: Post "https://metadata-store.tap.tanzu.example.com/api/sourceReport?": tls:
failed to verify certificate: x509: certificate signed by unknown authority
```

To ensure that the `caSecret` from the scanner `DEV-NAMESPACE` matches the `caSecret` from the `METADATASTORE-NAMESPACE` namespace:

1. In a single cluster, the connection between the scanning pod and the metadata store happens inside the cluster and does not pass through ingress. This is automatically configured. You do not need to provide an ingress connection to the store. If you provided an ingress connection to the store, delete it.
2. Get the `caSecret.name` depending if your setup is single or multicluster.
    1. If you are using a single cluster setup, the default value for `grype.metadataStore.caSecret.name` is `app-tls-cert`. See Install Supply Chain Security Tools - Scan.
    2. If you are using a multicluster setup, retrieve `grype.metadataStore.caSecret.name` from the Grype config:

        ```
        grype:
        metadataStore:
          caSecret:
            name: store-ca-cert
            importFromNamespace: metadata-store-secrets
        ```

        Note `caSecret.name` is set to `store-ca-cert`. See Multicluster setup.
3. Verify that the `CA-SECRET` secret exists in the `DEV-NAMESPACE`.

    ```
    kubectl get secret CA-SECRET -n DEV-NAMESPACE
    ```
4. If the secret `CA-SECRET` doesn't exist in your `DEV-NAMESPACE`, verify that the `CA-SECRET` exists in the `METADATASTORE-NAMESPACE` namespace:

    ```
    kubectl get secret CA-SECRET -n METADATASTORE-NAMESPACE
    ```

Where `METADATASTORE-NAMESPACE` is the namespace that contains the secret `CA-SECRET`. If you are using a single cluster, it is configured using the `metadata-store` namespace. If multicluster, it is configured using the `metadata-store-secrets`.

- If `CA-SECRET` doesn't exist in the metadata store namespace, configure the certificate. See Custom certificate configuration.

5. Check if the secretexport and secretimport exist and are reconciling successfully:

```
kubectl get secretexports.secretgen.carvel.dev -n `METADATASTORE-NAMESPACE`
kubectl get secretimports.secretgen.carvel.dev -n `DEV-NAMESPACE`
```

6. Check that the `ca.crt` field in both secrets from `METADATASTORE-NAMESPACE` and `DEV-NAMESPACE` match, or that the `ca.crt` field of the secret in the `METADATASTORE-NAMESPACE` includes the `ca.crt` field of the secret from the `DEV-NAMESPACE`.

   Confirm this by base64 decoding both secrets and verifying that there is a match:

```
kubectl get secret CA-SECRET -n DEV-NAMESPACE -o json | jq -r '.data."ca.crt"'
| base64 -d
kubectl get secret CA-SECRET -n METADATASTORE-NAMESPACE -o json | jq -r '.dat
a."ca.crt"' | base64 -d
```

   The certificates in the `METADATASTORE-NAMESPACE` and `DEV-NAMESPACE` must have a match for the scanner to connect to the metadata-store.

Troubleshooting issues

Troubleshooting Grype in air gap Environments

For information about issues with Grype in air gap environments, see Using Grype in offline and air-gapped environments.

Missing target SSH secret

Scanning source code from a private source repository requires an SSH secret present in the namespace and referenced as `grype.targetSourceSshSecret` in `tap-values.yaml`. See Installing the Tanzu Application Platform Package and Profiles.

If a private source scan is triggered and the secret cannot be found, the scan pod includes a `FailedMount` warning in Events with the message `MountVolume.SetUp failed for volume "ssh-secret" : secret "secret-ssh-auth" not found`, where `secret-ssh-auth` is the value specified in `grype.targetSourceSshSecret`.

Missing target image pull secret

Scanning an image from a private registry requires an image pull secret to exist in the Scan CRs namespace and be referenced as `grype.targetImagePullSecret` in `tap-values.yaml`. See Installing the Tanzu Application Platform Package and Profiles.

If a private image scan is triggered and the secret is not configured, the scan TaskRun's pod's `step-scan-plugin` container fails with the following error:

```
Error: GET https://dev.registry.tanzu.vmware.com/v2/vse-dev/spring-petclinic/manifest
s/sha256:128e38c1d3f10401a595c253743bee343967c81e8f22b94e30b2ab8292b3973f: UNAUTHORIZE
D: unauthorized to access repository: vse-dev/spring-petclinic, action: pull: unauthor
ized to access repository: vse-dev/spring-petclinic, action: pull
```

Deactivate Supply Chain Security Tools (SCST) - Store

SCST - Store is required to install SCST - Scan. If you install without the SCST - Store, you must edit the configurations to deactivate the Store:

```
---
metadataStore:
  url: ""
```

Install the package with the edited configurations by running:

```
tanzu package install scan-controller \
  --package scanning.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file tap-values.yaml
```

Resolving Incompatible Syft Schema Version

You might encounter the following error:

```
The provided SBOM has a Syft Schema Version which doesn't match the version that is su
pported by Grype...
```

This means that the Syft Schema Version from the provided SBOM doesn't match the version supported by the installed `grype-scanner`. There are two different methods to resolve this incompatibility issue:

- (Preferred method) Install a version of Tanzu Build Service that provides an SBOM with a compatible Syft Schema Version.
- Deactivate the `failOnSchemaErrors` in `grype-values.yaml`. See Install Supply Chain Security Tools - Scan. Although this change bypasses the check on Syft Schema Version, it does not resolve the incompatibility issue and produces a partial scanning result.

  ```
  syft:
    failOnSchemaErrors: false
  ```

Resolving incompatible scan policy

If your scan policy appears to not be enforced, it might be because the Rego file defined in the scan policy is incompatible with the scanner that is being used. For example, the Grype Scanner outputs in the CycloneDX XML format while the Snyk Scanner outputs SPDX JSON.

See Sample ScanPolicy for Snyk in SPDX JSON format for an example of a ScanPolicy formatted for SPDX JSON.

Could not find CA in secret

If you encounter the following issue, it might be due to not exporting `app-tls-cert` to the correct namespace:

```
{"level":"error","ts":"2022-06-08T15:20:48.43237873Z","logger":"setup","msg":"Could no
t find CA in Secret","err":"unable to set up connection to Supply Chain Security Tools
- Store"}
```

Configure `ns_for_export_app_cert` in your `tap-values.yaml`.

```
metadata_store:
  ns_for_export_app_cert: "DEV-NAMESPACE"
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

If there are multiple developer namespaces, use `ns_for_export_app_cert: "*"`.

Blob Source Scan is reporting wrong source URL

A Source Scan for a blob artifact can cause reporting in the `status.artifact` and `status.compliantArtifact` the wrong URL for the resource, passing the remote SSH URL instead of the cluster local fluxcd one. One symptom of this issue is the `image-builder` failing with a `ssh:// is an unsupported protocol` error message.

You can confirm you're having this problem by running `kubectl describe` in the affected resource and comparing the `spec.blob.url` value against the `status.artifact.blob.url`. The problem occurs if they are different URLs. For example:

```
kubectl describe sourcescan SOURCE-SCAN-NAME -n DEV-NAMESPACE
```

Where:

- `SOURCE-SCAN-NAME` is the name of the source scan you want to configure.
- `DEV-NAMESPACE` is the name of the developer namespace you want to use. And compare the output:

```
...
spec:
  blob:
    ...
    url: http://source-controller.flux-system.svc.cluster.local./gitrepository/sample/
repo/8d4cea98b0fa9e0112d58414099d0229f190f7f1.tar.gz
    ...
status:
  artifact:
    blob:
      ...
      url: ssh://git@github.com:sample/repo.git
  compliantArtifact:
    blob:
      ...
      url: ssh://git@github.com:sample/repo.git
```

Workaround: This problem happens in SCST - Scan `v1.2.0` when you use a Grype Scanner ScanTemplates earlier than `v1.2.0`, because this is a deprecated path. To fix this problem, upgrade your Grype Scanner deployment to `v1.2.0` or later. See Upgrading Supply Chain Security Tools - Scan for step-by-step instructions.

Resolving failing scans that block a Supply Chain

If the Supply Chain is not progressing due to CVEs found in either the SourceScan or ImageScan, see the CVE triage workflow in Triaging and Remediating CVEs.

Policy not defined in the Tanzu Application Platform GUI

If you encounter `No policy has been defined`, it might be because the Tanzu Application Platform GUI is unable to view the Scan Policy resource.

Confirm that the Scan Policy associated with a SourceScan or ImageScan exists. For example, the `scanPolicy` in the scan matches the name of the Scan Policy.

```
kubectl describe sourcescan NAME -n DEV-NAMESPACE
kubectl describe imagescan NAME -n DEV-NAMESPACE
```

```
kubectl get scanpolicy NAME -n DEV-NAMESPACE
```

Where `DEV-NAMESPACE` is the name of the developer namespace you want to use.

Add the `app.kubernetes.io/part-of` label to the Scan Policy. See Enable Tanzu Application Platform GUI to view ScanPolicy Resource.

Lookup error when connecting to SCST - Store

If your scan pod is failing, you might see the following connection error in the logs:

```
dial tcp: lookup metadata-store-app.metadata-store.svc.cluster.local on 10.100.0.10:5
3: no such host
```

A connection error while attempting to connect to the local cluster URL causes this error. If this is a multicluster deployment, set the `grype.metadataStore.url` property in your Build profile `values.yaml`. You must set the ingress domain of SCST - Store which is deployed in the View cluster. For information about this configuration, see Install Build profile.

Sourcescan error with SCST - Store endpoint without a prefix

If your Source Scan resource is failing, the status might show this error:

```
Error: endpoint require 'http://' or 'https://' prefix
```

This is because the `grype.metadataStore.url` value in the Tanzu Application Platform profile `values.yaml` was not configured with the correct prefix. Verify that the URL starts with either `http://` or `https://`.

Deprecated pre-v1.2 templates

If the scan phase is in `Error` and the status condition message is:

```
Summary logs could not be retrieved: . error opening stream pod logs reader: container
summary is not valid for pod scan-grypeimagescan-sample-public-zmj2g-hqv5g
```

This error might be a consequence of using Grype Scanner ScanTemplates shipped with SCST - Scan v1.1 or earlier. These ScanTemplates are deprecated and are not supported in Tanzu Application Platform v1.4.0 and later.

There are two options to resolve this issue:

- Option 1: Upgrade to the latest Grype Scanner version. This automatically replaces the old ScanTemplates with the upgraded ScanTemplates.
- Option 2: Create a ScanTemplate. Follow the steps in Create a scan template.

Incorrectly configured self-signed certificate

The following error in the pod logs indicate that the self-signed certificate might be incorrectly configured:

```
x509: certificate signed by unknown authority
```

To resolve this issue, ensure that `shared.ca_cert_data` contains the required certificate. For an example of setting up the shared self-signed certificate, see Build profile.

For information about `shared.ca_cert_data`, see View possible configuration settings for your package.

Unable to pull scan controller and scanner images from a specified registry

The `docker` field and related sub-fields by SCST - Scan Controller, Grype Scanner, or Snyk Scanner are deprecated in Tanzu Application Platform v1.4.0. Previously these text boxes might be used to populate the `registry-credentials` secret. You might encounter the following error during installation:

```
UNAUTHORIZED: unauthorized to access repository
```

The recommended migration path for users setting up their namespaces manually is to add registry credentials to both the developer namespace and the `scan-link-system` namespace, using these instructions.

Important

This step does not apply to users who used `--export-to-all-namespaces` when setting up the Tanzu Application Platform package repository.

Grype database not available

Before running a scan, the Grype scanner downloads a copy of its database. If the database fails to download, the following log entry might appear.

```
Vulnerability DB [no update available] New version of grype is available: 0.50.2 [000
0] WARN unable to check for vulnerability database update 1 error occurred: * failed t
o load vulnerability db: vulnerability database is corrupt (run db update to correct):
database metadata not found: ~/Library/Caches/grype/db/3
```

To resolve this issue, ensure that Grype has access to its vulnerability database:

- If you have set up a mirror of the vulnerability database, verify that it is populated and reachable.
- If you did not set up a mirror, Grype manages its database behind the scenes. Verify that the cluster has access to https://anchore.com/.

This issue is unrelated to Supply Chain Security Tools for Tanzu – Store.

Scanner Pod restarts once in SCST - Scan `v1.5.0` or later

For SCST - Scan `v1.5.0` or later, you see scanner pods restart:

```
Pods
   NAME                                READY   STATUS      RESTARTS   AGE
   my-scan-45smk-pod                   0/9     Completed   1          14m
```

One restart in scanner pods is expected with successful scans. To support Tanzu Service Mesh (TSM) integration, jobs were replaced with TaskRuns. This restart is an artifact of how Tekton cleans up sidecar containers by patching the container specifications.

Troubleshoot Rego files with a scan policy for Supply Chain Security Tools - Scan

This topic describes how you can use an example output to troubleshoot your Rego file for SCST - Scan. You use a Rego file in a scan policy custom resource. See Enforce compliance policy using Open Policy Agent.

For information about how to write Rego, see Open Policy Agent documentation.

Using the Rego playground

Use the Rego Playground, to evaluate your Rego file against an input. In this example, use the example output of an image or source scan custom resource.

Sample input in CycloneDX's XML re-encoded as JSON format

The following is an example scan custom resource output in CycloneDX's XML structure re-encoded as JSON. This example output contains CVEs at low, medium, high, and critical severities.

To troubleshoot using this example output:

1. Paste your Rego file and the example output into the Rego Playground.
2. Evaluate your Rego file against the example output and verify that your Rego file detects the intended CVEs. See this Rego example.


```
{
    "bom": {
        "-serialNumber": "urn:uuid:123",
        "-v": "http://cyclonedx.org/schema/ext/vulnerability/1.0",
        "-version": "1",
        "-xmlns": "http://cyclonedx.org/schema/bom/1.2",
        "components": {
            "component": [
                {
                    "-type": "library",
                    "licenses": {
                        "license": {
                            "name": "GPL-2"
                        }
                    },
                    "name": "adduser",
                    "version": "3.118",
                    "vulnerabilities": {
                        "vulnerability": [
                            {
                                "-ref": "urn:uuid:3d7c61c6-9cfa-494c-858a-9668a318ff2
3",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2011-3374"
                                },
                                "id": "CVE-2011-3374-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Low"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2011-3374"
                                }
                            },
                            {
                                "-ref": "urn:uuid:ebabaa92-2bf9-4d33-8181-595b0fdf55b
d",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-27350"
                                },
```

```
                                "id": "CVE-2020-27350-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Medium"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-27350"
                                }
                            },
                            {
                                "-ref": "urn:uuid:07c58c81-1e01-459d-9e9d-0e10456a9bf
0",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-3810"
                                },
                                "id": "CVE-2020-3810-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Medium"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-3810"
                                }
                            }
                        ]
                    }
                },
                {
                    "-type": "library",
                    "licenses": {
                        "license": [
                            {
                                "name": "GPL-2"
                            },
                            {
                                "name": "GPLv2+"
                            }
                        ]
                    },
                    "name": "apt",
                    "version": "1.8.2",
                    "vulnerabilities": {
                        "vulnerability": [
                            {
                                "-ref": "urn:uuid:3d7c61c6-9cfa-494c-858a-9668a318ff2
3",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2011-3374"
                                },
                                "id": "CVE-2011-3374",
                                "ratings": {
                                    "rating": {
                                        "severity": "Low"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
```

```
ame=CVE-2011-3374"
                                }
                            },
                            {
                                "-ref": "urn:uuid:ebabaa92-2bf9-4d33-8181-595b0fdf55b
d",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-27350"
                                },
                                "id": "CVE-2020-27350",
                                "ratings": {
                                    "rating": {
                                        "severity": "High"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-27350"
                                }
                            },
                            {
                                "-ref": "urn:uuid:07c58c81-1e01-459d-9e9d-0e10456a9bf
0",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-3810"
                                },
                                "id": "CVE-2020-3810",
                                "ratings": {
                                    "rating": {
                                        "severity": "Critical"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-3810"
                                }
                            }
                        ]
                    }
                }
            ]
        },
        "metadata": {
            "component": {
                "-type": "container",
                "name": "nginx:1.16",
                "version": "sha256:123"
            },
            "timestamp": "2022-01-28T13:30:43-08:00",
            "tools": {
                "tool": {
                    "name": "grype",
                    "vendor": "anchore",
                    "version": "[not provided]"
                }
            }
        }
    }
}
```


Example input in SPDX JSON format

The example in this section is a modified scan custom resource input, in `.spdx.json`, that contains CVEs at low, medium, high, and critical severities. You can use this example input to evaluate your Rego file.

To troubleshoot using this example output:

1. Paste your Rego file and the example input into the Rego Playground.
2. Evaluate your Rego file against the output and verify that your Rego file detects the intended CVEs. See this Rego example.


```
{
  "id": "SPDXRef-docker-image|nginx",
  "specVersion": "SPDX-3.0",
  "creator": "Organization: Snyk Ltd",
  "created": "2023-03-01T16:10:08Z",
  "profile": [
    "base",
    "vulnerabilities"
  ],
  "description": "Snyk test result for project docker-image|nginx in SPDX SBOM forma
t",
  "vulnerabilities": [
    {
      "id": "SNYK-DEBIAN10-APT-1049974",
      "name": "SNYK-DEBIAN10-APT-1049974",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nAPT had several integer overflows and underflows while parsing .deb packages, a
ka GHSL-2020-168 GHSL-2020-169, in files apt-pkg/contrib/extracttar.cc, apt-pkg/deb/de
bfile.cc, and apt-pkg/contrib/arfile.cc. This issue affects: apt 1.2.32ubuntu0 version
s prior to 1.2.32ubuntu0.2; 1.6.12ubuntu0 versions prior to 1.6.12ubuntu0.2; 2.0.2ubun
tu0 versions prior to 2.0.2ubuntu0.2; 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.
1;\n## Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.2 or higher.\n## Refere
nces\n- [ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-27350)\n- [CON
FIRM](https://bugs.launchpad.net/bugs/1899193)\n- [CONFIRM](https://security.netapp.co
m/advisory/ntap-20210108-0005/)\n- [DEBIAN](https://www.debian.org/security/2020/dsa-4
808)\n- [UBUNTU](https://usn.ubuntu.com/usn/usn-4667-1)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
```

```
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.7
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:L/I:L/A:L"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-27350"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1899193"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20210108-0005/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2020/dsa-4808"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/usn/usn-4667-1"
        }
      ],
      "modified": "2022-10-29T13:11:02.438923Z",
      "published": "2020-12-10T03:10:23.901831Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-407502",
      "name": "SNYK-DEBIAN10-APT-407502",
      "summary": "Improper Verification of Cryptographic Signature",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nIt was found that apt-key in apt, all versions, do not correctly validate gpg k
eys with the primary keyring, leading to a potential man-in-the-middle attack.\n## Rem
ediation\nThere is no fixed version for `Debian:10` `apt`.\n## References\n- [ADVISOR
Y](https://security-tracker.debian.org/tracker/CVE-2011-3374)\n- [Debian Bug Report](h
ttps://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480)\n- [MISC](https://people.cano
nical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html)\n- [MISC](https://seclists.or
g/fulldisclosure/2011/Sep/221)\n- [MISC](https://snyk.io/vuln/SNYK-LINUX-APT-116518)\n
- [MISC](https://ubuntu.com/security/CVE-2011-3374)\n- [RedHat CVE Database](https://a
ccess.redhat.com/security/cve/cve-2011-3374)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
```

```
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              347
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 3.7
                  }
                ],
                "severity": "Low",
                "vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480"
        },
        {
          "category": "ADVISORY",
          "locator": "https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-
3374.html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://seclists.org/fulldisclosure/2011/Sep/221"
        },
        {
          "category": "ADVISORY",
          "locator": "https://snyk.io/vuln/SNYK-LINUX-APT-116518"
        },
        {
          "category": "ADVISORY",
          "locator": "https://ubuntu.com/security/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://access.redhat.com/security/cve/cve-2011-3374"
```

```
        }
      ],
      "modified": "2022-11-01T00:08:27.375895Z",
      "published": "2018-06-27T16:20:45.037549Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-568926",
      "name": "SNYK-DEBIAN10-APT-568926",
      "summary": "Improper Input Validation",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nMissing input validation in the ar/tar implementations of APT before version 2.
1.2 could result in denial of service when processing specially crafted deb files.\n##
Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.1 or higher.\n## References\n-
[ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-3810)\n- [FEDORA](http
s://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/mes
sage/U4PEH357MZM2SUGKETMEHMSGQS652QHH/)\n- [GitHub Issue](https://github.com/Debian/ap
t/issues/111)\n- [MISC](https://bugs.launchpad.net/bugs/1878177)\n- [MISC](https://lis
ts.debian.org/debian-security-announce/2020/msg00089.html)\n- [MISC](https://salsa.deb
ian.org/apt-team/apt/-/commit/dceb1e49e4b8e4dadaf056be34088b415939cda6)\n- [MISC](http
s://tracker.debian.org/news/1144109/accepted-apt-212-source-into-unstable/)\n- [UBUNT
U](https://usn.ubuntu.com/4359-2/)\n- [Ubuntu CVE Tracker](http://people.ubuntu.com/~u
buntu-security/cve/CVE-2020-3810)\n- [Ubuntu Security Advisory](https://usn.ubuntu.co
m/4359-1/)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              20
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.5
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:H"
              }
            ]
```

```
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-3810"
        },
        {
          "category": "ADVISORY",
          "locator": "https://lists.fedoraproject.org/archives/list/package-announce@l
ists.fedoraproject.org/message/U4PEH357MZM2SUGKETMEHMSGQS652QHH/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/Debian/apt/issues/111"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1878177"
        },
        {
          "category": "ADVISORY",
          "locator": "https://lists.debian.org/debian-security-announce/2020/msg00089.
html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://salsa.debian.org/apt-team/apt/-/commit/dceb1e49e4b8e4dad
af056be34088b415939cda6"
        },
        {
          "category": "ADVISORY",
          "locator": "https://tracker.debian.org/news/1144109/accepted-apt-212-source-
into-unstable/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/4359-2/"
        },
        {
          "category": "ADVISORY",
          "locator": "http://people.ubuntu.com/~ubuntu-security/cve/CVE-2020-3810"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/4359-1/"
        }
      ],
      "modified": "2022-11-01T00:08:51.907776Z",
      "published": "2020-05-12T14:19:01.052295Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-1049974",
      "name": "SNYK-DEBIAN10-APT-1049974",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nAPT had several integer overflows and underflows while parsing .deb packages, a
ka GHSL-2020-168 GHSL-2020-169, in files apt-pkg/contrib/extracttar.cc, apt-pkg/deb/de
bfile.cc, and apt-pkg/contrib/arfile.cc. This issue affects: apt 1.2.32ubuntu0 version
s prior to 1.2.32ubuntu0.2; 1.6.12ubuntu0 versions prior to 1.6.12ubuntu0.2; 2.0.2ubun
tu0 versions prior to 2.0.2ubuntu0.2; 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.
1;\n## Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.2 or higher.\n## Refere
nces\n- [ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-27350)\n- [CON
```

```
FIRM](https://bugs.launchpad.net/bugs/1899193)\n- [CONFIRM](https://security.netapp.co
m/advisory/ntap-20210108-0005/)\n- [DEBIAN](https://www.debian.org/security/2020/dsa-4
808)\n- [UBUNTU](https://usn.ubuntu.com/usn/usn-4667-1)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt@1.8.2",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.7
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:L/I:L/A:L"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-27350"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1899193"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20210108-0005/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2020/dsa-4808"
        },
        {
```

```
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/usn/usn-4667-1"
        }
      ],
      "modified": "2022-10-29T13:11:02.438923Z",
      "published": "2020-12-10T03:10:23.901831Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-407502",
      "name": "SNYK-DEBIAN10-APT-407502",
      "summary": "Improper Verification of Cryptographic Signature",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nIt was found that apt-key in apt, all versions, do not correctly validate gpg k
eys with the primary keyring, leading to a potential man-in-the-middle attack.\n## Rem
ediation\nThere is no fixed version for `Debian:10` `apt`.\n## References\n- [ADVISOR
Y](https://security-tracker.debian.org/tracker/CVE-2011-3374)\n- [Debian Bug Report](h
ttps://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480)\n- [MISC](https://people.cano
nical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html)\n- [MISC](https://seclists.or
g/fulldisclosure/2011/Sep/221)\n- [MISC](https://snyk.io/vuln/SNYK-LINUX-APT-116518)\n
- [MISC](https://ubuntu.com/security/CVE-2011-3374)\n- [RedHat CVE Database](https://a
ccess.redhat.com/security/cve/cve-2011-3374)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt@1.8.2",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              347
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 3.7
                  }
                ],
                "severity": "Low",
                "vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N"
              }
            ]
          }
```

```
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480"
        },
        {
          "category": "ADVISORY",
          "locator": "https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-
3374.html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://seclists.org/fulldisclosure/2011/Sep/221"
        },
        {
          "category": "ADVISORY",
          "locator": "https://snyk.io/vuln/SNYK-LINUX-APT-116518"
        },
        {
          "category": "ADVISORY",
          "locator": "https://ubuntu.com/security/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://access.redhat.com/security/cve/cve-2011-3374"
        }
      ],
      "modified": "2022-11-01T00:08:27.375895Z",
      "published": "2018-06-27T16:20:45.037549Z"
    },
    {
      "id": "SNYK-DEBIAN10-EXPAT-2329087",
      "name": "SNYK-DEBIAN10-EXPAT-2329087",
      "summary": "Incorrect Calculation",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `expat` package and not the `expat` package as distribut
ed by `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and st
atus._\n\nIn Expat (aka libexpat) before 2.4.3, a left shift by 29 (or more) places in
the storeAtts function in xmlparse.c can lead to realloc misbehavior (e.g., allocating
too few bytes, or only freeing memory).\n## Remediation\nUpgrade `Debian:10` `expat` t
o version 2.2.6-2+deb10u2 or higher.\n## References\n- [ADVISORY](https://security-tra
cker.debian.org/tracker/CVE-2021-45960)\n- [MISC](https://bugzilla.mozilla.org/show_bu
g.cgi?id=1217609)\n- [MISC](https://github.com/libexpat/libexpat/issues/531)\n- [MISC]
(https://github.com/libexpat/libexpat/pull/534)\n- [cve@mitre.org](http://www.openwal
l.com/lists/oss-security/2022/01/17/3)\n- [cve@mitre.org](https://security.netapp.com/
advisory/ntap-20220121-0004/)\n- [cve@mitre.org](https://www.tenable.com/security/tns-
2022-05)\n- [cve@mitre.org](https://www.debian.org/security/2022/dsa-5073)\n- [cve@mit
re.org](https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf)\n- [cve@mitre.
org](https://security.gentoo.org/glsa/202209-24)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "nginx-module-image-filter@1.16.1-1~buster",
              "libgd2/libgd3@2.2.5-5.2",
              "fontconfig/libfontconfig1@2.13.1-2",
              "expat/libexpat1@2.2.6-2+deb10u1"
            ],
            "type": "AFFECTS"
```

```
        },
        "foundBy": {
          "to": [
            ""
          ],
          "type": "FOUND_BY"
        },
        "suppliedBy": {
          "to": [
            ""
          ],
          "type": "SUPPLIED_BY"
        },
        "ratedBy": {
          "to": [
            ""
          ],
          "type": "RATED_BY",
          "cwes": [
            682
          ],
          "rating": [
            {
              "method": "CVSS_3",
              "score": [
                {
                  "base": 8.8
                }
              ],
              "severity": "High",
              "vector": "CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H"
            }
          ]
        }
      }
    ],
    "externalReferences": [
      {
        "category": "ADVISORY",
        "locator": "https://security-tracker.debian.org/tracker/CVE-2021-45960"
      },
      {
        "category": "ADVISORY",
        "locator": "https://bugzilla.mozilla.org/show_bug.cgi?id=1217609"
      },
      {
        "category": "ADVISORY",
        "locator": "https://github.com/libexpat/libexpat/issues/531"
      },
      {
        "category": "ADVISORY",
        "locator": "https://github.com/libexpat/libexpat/pull/534"
      },
      {
        "category": "ADVISORY",
        "locator": "http://www.openwall.com/lists/oss-security/2022/01/17/3"
      },
      {
        "category": "ADVISORY",
        "locator": "https://security.netapp.com/advisory/ntap-20220121-0004/"
      },
      {
        "category": "ADVISORY",
        "locator": "https://www.tenable.com/security/tns-2022-05"
      },
      {
```

```
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2022/dsa-5073"
        },
        {
          "category": "ADVISORY",
          "locator": "https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.gentoo.org/glsa/202209-24"
        }
      ],
      "modified": "2023-02-14T13:37:37.505975Z",
      "published": "2022-01-02T01:41:26.770663Z"
    },
    {
      "id": "SNYK-DEBIAN10-EXPAT-2331803",
      "name": "SNYK-DEBIAN10-EXPAT-2331803",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `expat` package and not the `expat` package as distribut
ed by `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and st
atus._\n\ndefineAttribute in xmlparse.c in Expat (aka libexpat) before 2.4.3 has an in
teger overflow.\n## Remediation\nUpgrade `Debian:10` `expat` to version 2.2.6-2+deb10u
2 or higher.\n## References\n- [ADVISORY](https://security-tracker.debian.org/tracker/
CVE-2022-22824)\n- [cve@mitre.org](https://github.com/libexpat/libexpat/pull/539)\n-
[cve@mitre.org](http://www.openwall.com/lists/oss-security/2022/01/17/3)\n- [cve@mitr
e.org](https://www.tenable.com/security/tns-2022-05)\n- [cve@mitre.org](https://www.de
bian.org/security/2022/dsa-5073)\n- [cve@mitre.org](https://cert-portal.siemens.com/pr
oductcert/pdf/ssa-484086.pdf)\n- [cve@mitre.org](https://security.gentoo.org/glsa/2022
09-24)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "nginx-module-image-filter@1.16.1-1~buster",
              "libgd2/libgd3@2.2.5-5.2",
              "fontconfig/libfontconfig1@2.13.1-2",
              "expat/libexpat1@2.2.6-2+deb10u1"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
```

```
                "score": [
                  {
                    "base": 9.8
                  }
                ],
                "severity": "Critical",
                "vector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2022-22824"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/pull/539"
        },
        {
          "category": "ADVISORY",
          "locator": "http://www.openwall.com/lists/oss-security/2022/01/17/3"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.tenable.com/security/tns-2022-05"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2022/dsa-5073"
        },
        {
          "category": "ADVISORY",
          "locator": "https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.gentoo.org/glsa/202209-24"
        }
      ],
      "modified": "2023-02-14T13:39:18.516672Z",
      "published": "2022-01-08T13:52:14.479733Z"
    }
  ],
  "name": "docker-image|nginx-sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5e
eb69fcaaf30dd9c",
  "dataLicense": "CC0-1.0",
  "documentNamespace": "spdx.org/spdxdocs/docker-image|nginx-feb02ce6-cd47-49c2-9a97-2
b4833b4a1f0",
  "relationships": [
    {
      "from": "SPDXRef-docker-image|nginx",
      "to": [
        "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4807e0de4
62caf2336f0b70b5eeb69fcaaf30dd9c"
      ],
      "type": "DESCRIBES"
    }
  ],
  "packages": [
    {
      "SPDXID": "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4
807e0de462caf2336f0b70b5eeb69fcaaf30dd9c",
      "versionInfo": "sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaa
```

```
f30dd9c",
      "id": "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4807e
0de462caf2336f0b70b5eeb69fcaaf30dd9c",
      "name": "index.docker.io/library/nginx",
      "checksums": [
        {
          "algorithm": "SHA256",
          "checksumValue": "d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaaf
30dd9c"
        }
      ]
    }
  ]
}
```

Configure code repositories and image artifacts for Supply Chain Security Tools - Scan

This topic describes how you can configure code repositories and image artifacts for SCST - Scan.

Prerequisite

Both the source and image scans require you to define a `ScanTemplate`. Run `kubectl get scantemplates` for the ScanTemplates provided with the scanner installation. For information about how to reference these ScanTemplates, see How to create a ScanTemplate.

Deploy scan custom resources

The scan controller defines two custom resources to create scans:

- SourceScan
- ImageScan

SourceScan

The `SourceScan` custom resource helps you define and trigger a scan for a given repository. You can deploy `SourceScan` with source code existing in a public repository or a private one:

1. Create the `SourceScan` custom resource.

   Example:

   ```
   apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
   kind: SourceScan
   metadata:
     # set the name of the source scan CR
     name: sample-source-scan
   spec:
     # At least one of these fields (blob or git) must be defined.
     blob:
       # location to a file with the source code compressed (supported files: .ta
   r.gz)
       url:
     git:
       # A multiline string defining the known hosts that are going to be used for
   the SSH client on the container
       knownHosts:
       # Branch, tag, or commit digest
       revision:
       # The name of the kubernetes secret containing the private SSH key informat
   ion.
   ```

```
    sshKeySecret:
    # A string containing the repository URL.
    url:
    # The username needed to SSH connection. Default value is “git”
    username:

  # A string defining the name of an existing ScanTemplate custom resource.
  scanTemplate: my-scan-template

   # A string defining the name of an existing ScanPolicy custom resource. See
"Enforcement Policies (OPA)" section.
  scanPolicy: my-scan-policy
```

2. Deploy the `SourceScan` custom resource to the desired namespace on cluster by running:

```
kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
amespace>
```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```
# These fields are populated from the source scan results
status:
  # The source code information as provided in the CycloneDX `bom>metadata>comp
onent>*` fields
  artifact:
    blob:
      url:
    git:
      url:
      revision:

  # An array populated with information about the scanning status
  # and the policy validation. These conditions might change in the lifecycle
  # of the scan, refer to the "View Scan Status and Understanding Conditions" s
ection to learn more.
  conditions: []

  # The URL of the vulnerability scan results in the Metadata Store integratio
n.
  # Only available when the integration is configured.
  metadataUrl:

  # When the CRD is updated to point at new revisions, this lets you know
  # if the status reflects the latest one or not
  observedGeneration: 1
  observedPolicyGeneration: 1
  observedTemplateGeneration: 1

  # The latest datetime when the scanning was successfully finished.
  scannedAt:
  # Information about the scanner that was used for the latest image scan.
  # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
  scannedBy:
    scanner:
      # The name of the scanner that was used.
      name: my-image-scanner

      # The name of the scanner's development company or team
      vendor: my-image-scanner-provider

      # The version of the scanner used.
      version: 1.0.0
```

ImageScan

The `ImageScan` custom resource helps you define and trigger a scan for a given image. You can deploy `ImageScan` with an image existing in a public or private registry:

1. Create the `ImageScan` custom resource.

   Example:

   ```
   apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
   kind: ImageScan
   metadata:
     # set the name of the image scan CR
     name: sample-image-scan
   spec:
     registry:
       # Required. A string containing the image name can additionally add its tag
   or its digest
       image: nginx:1.16

       # A string containing the secret needed to pull the image from a private re
   gistry.
       # The secret needs to be deployed in the same namespace as the ImageScan
       imagePullSecret: my-image-pull-secret

     # A string defining the name of an existing ScanTemplate custom resource. See
   "How To Create a ScanTemplate" section.
     scanTemplate: my-scan-template

     # A string defining the name of an existing ScanPolicy custom resource. See
   "Enforcement Policies (OPA)" section.
     scanPolicy: my-scan-policy
   ```

2. Deploy the `ImageScan` custom resource to the desired namespace on cluster by running:

   ```
   kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
   amespace>
   ```

   After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

   ```
    # These fields are populated from the image scan results
   status:
     artifact:
       registry:
         # The image name with its digest as provided in the CycloneDX `bom>metada
   ta>component>*` fields
         image:
         imagePullSecret:

     # An array that is populated with information about the scanning status
     # and the policy validation. These conditions might change in the lifecycle
     # of the scan, refer to the "View Scan Status and Understanding Conditions" s
   ection to learn more.
     conditions: []

     # The URL of the vulnerability scan results in the Metadata Store integratio
   n.
     # Only available when the integration is configured.
     metadataUrl:

     # When the CRD is updated to point at new revisions, this lets you know
     # whether the status reflects the latest one
     observedGeneration: 1
     observedPolicyGeneration: 1
   ```

```
  observedTemplateGeneration: 1

  # The latest datetime when the scanning was successfully finished.
  scannedAt:
  # Information about the scanner used for the latest image scan.
  # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
  scannedBy:
    scanner:
      # The name of the scanner that was used.
      name: my-image-scanner

      # The name of the scanner's development company or team
      vendor: my-image-scanner-provider

      # The version of the scanner used.
      version: 1.0.0
```

Configure code repositories and image artifacts for Supply Chain Security Tools - Scan

This topic describes how you can configure code repositories and image artifacts for SCST - Scan.

Prerequisite

Both the source and image scans require you to define a `ScanTemplate`. Run `kubectl get scantemplates` for the ScanTemplates provided with the scanner installation. For information about how to reference these ScanTemplates, see How to create a ScanTemplate.

Deploy scan custom resources

The scan controller defines two custom resources to create scans:

- SourceScan
- ImageScan

SourceScan

The `SourceScan` custom resource helps you define and trigger a scan for a given repository. You can deploy `SourceScan` with source code existing in a public repository or a private one:

1. Create the `SourceScan` custom resource.

   Example:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: SourceScan
metadata:
  # set the name of the source scan CR
  name: sample-source-scan
spec:
  # At least one of these fields (blob or git) must be defined.
  blob:
    # location to a file with the source code compressed (supported files: .ta
r.gz)
    url:
  git:
    # A multiline string defining the known hosts that are going to be used for
the SSH client on the container
    knownHosts:
    # Branch, tag, or commit digest
```

```
    revision:
    # The name of the kubernetes secret containing the private SSH key informat
ion.
    sshKeySecret:
    # A string containing the repository URL.
    url:
    # The username needed to SSH connection. Default value is “git”
    username:

  # A string defining the name of an existing ScanTemplate custom resource.
  scanTemplate: my-scan-template

   # A string defining the name of an existing ScanPolicy custom resource. See
"Enforcement Policies (OPA)" section.
  scanPolicy: my-scan-policy
```

2. Deploy the `SourceScan` custom resource to the desired namespace on cluster by running:

```
kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
amespace>
```

After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

```
# These fields are populated from the source scan results
status:
  # The source code information as provided in the CycloneDX `bom>metadata>comp
onent>*` fields
  artifact:
    blob:
      url:
    git:
      url:
      revision:

  # An array populated with information about the scanning status
  # and the policy validation. These conditions might change in the lifecycle
  # of the scan, refer to the "View Scan Status and Understanding Conditions" s
ection to learn more.
  conditions: []

  # The URL of the vulnerability scan results in the Metadata Store integratio
n.
  # Only available when the integration is configured.
  metadataUrl:

  # When the CRD is updated to point at new revisions, this lets you know
  # if the status reflects the latest one or not
  observedGeneration: 1
  observedPolicyGeneration: 1
  observedTemplateGeneration: 1

  # The latest datetime when the scanning was successfully finished.
  scannedAt:
  # Information about the scanner that was used for the latest image scan.
  # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
  scannedBy:
    scanner:
      # The name of the scanner that was used.
      name: my-image-scanner

      # The name of the scanner's development company or team
      vendor: my-image-scanner-provider
```

```
      # The version of the scanner used.
      version: 1.0.0
```

ImageScan

The `ImageScan` custom resource helps you define and trigger a scan for a given image. You can deploy `ImageScan` with an image existing in a public or private registry:

1. Create the `ImageScan` custom resource.

   Example:

   ```
   apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
   kind: ImageScan
   metadata:
     # set the name of the image scan CR
     name: sample-image-scan
   spec:
     registry:
       # Required. A string containing the image name can additionally add its tag
   or its digest
       image: nginx:1.16

       # A string containing the secret needed to pull the image from a private re
   gistry.
       # The secret needs to be deployed in the same namespace as the ImageScan
       imagePullSecret: my-image-pull-secret

     # A string defining the name of an existing ScanTemplate custom resource. See
   "How To Create a ScanTemplate" section.
     scanTemplate: my-scan-template

     # A string defining the name of an existing ScanPolicy custom resource. See
   "Enforcement Policies (OPA)" section.
     scanPolicy: my-scan-policy
   ```

2. Deploy the `ImageScan` custom resource to the desired namespace on cluster by running:

   ```
   kubectl apply -f <path_to_the_cr>/<custom_resource_filename>.yaml -n <desired_n
   amespace>
   ```

   After the scanning completes, the following fields appear in the custom resource and are filled by the scanner:

   ```
    # These fields are populated from the image scan results
   status:
     artifact:
       registry:
         # The image name with its digest as provided in the CycloneDX `bom>metada
   ta>component>*` fields
         image:
         imagePullSecret:

     # An array that is populated with information about the scanning status
     # and the policy validation. These conditions might change in the lifecycle
     # of the scan, refer to the "View Scan Status and Understanding Conditions" s
   ection to learn more.
     conditions: []

     # The URL of the vulnerability scan results in the Metadata Store integratio
   n.
     # Only available when the integration is configured.
     metadataUrl:
   ```

```
  # When the CRD is updated to point at new revisions, this lets you know
  # whether the status reflects the latest one
  observedGeneration: 1
  observedPolicyGeneration: 1
  observedTemplateGeneration: 1

  # The latest datetime when the scanning was successfully finished.
  scannedAt:
  # Information about the scanner used for the latest image scan.
  # This information reflects what's in the CycloneDX `bom>metadata>tools>tool>
*` fields.
  scannedBy:
    scanner:
      # The name of the scanner that was used.
      name: my-image-scanner

      # The name of the scanner's development company or team
      vendor: my-image-scanner-provider

      # The version of the scanner used.
      version: 1.0.0
```

Enforce compliance policy using Open Policy Agent

This topic describes how you can use Open Policy Agent to enforce compliance policy for Supply Chain Security Tools - Scan.

Writing a policy template

The Scan Policy custom resource (CR) allows you to define a Rego file for policy enforcement that you can reuse across image scan and source scan CRs.

The Scan Controller supports policy enforcement by using an Open Policy Agent (OPA) engine with Rego files. This allows you to validate scan results for company policy compliance and can prevent source code from being built or images from being deployed.

Rego file contract

To define a Rego file for an image scan or source scan, you must comply with the requirements defined for every Rego file for the policy verification to work properly. See Open Policy Agent documentation on how to write Rego.

- **Package main:** The Rego file must define a package in its body called `main`. The system looks for this package to verify the scan results compliance.
- **Input match:** The Rego file evaluates one vulnerability match at a time, iterating as many times as the Rego file finds vulnerabilities in the scan. The match structure is accessed in the `input.currentVulnerability` object inside the Rego file and has the CycloneDX format.
- **deny rule:** The Rego file must define a `deny` rule inside its body. `deny` is a set of error messages that are returned to the user. Each rule you write adds to that set of error messages. If the conditions in the body of the `deny` statement are true then the user is handed an error message. If false, the vulnerability is allowed in the Source or Image scan.

Define a Rego file for policy enforcement

Follow these steps to define a Rego file for policy enforcement that you can reuse across image scan and source scan CRs that output in the CycloneDX XML format.

Note

The Snyk Scanner outputs SPDX JSON. For an example of a ScanPolicy formatted for SPDX JSON output, see Sample ScanPolicy for Snyk in SPDX JSON format.

1. Create a scan policy with a Rego file. The following is an example scan policy resource:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "Unkn
ownSeverity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
    ignoreCves := []

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := mat
ch.ratings.rating[_].severity }
      some i
      fails := contains(notAllowedSeverities, severities[i])
      not fails
    }

    isSafe(match) {
      ignore := contains(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.b
om.components.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := com
p.vulnerabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ra
tings.rating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

You can edit the following text boxes of the Rego file as part of the CVE triage workflow:

- `notAllowedSeverities` contains the categories of CVEs that cause the SourceScan or ImageScan failing policy enforcement. The following example shows an `app-operator` blocking only `Critical`, `High` and `UnknownSeverity` CVEs.

  ```
  ...
  spec:
  ```

```
regoFile: |
  package main

  # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible",
"UnknownSeverity"
  notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]
  ignoreCves := []
...
```

- `ignoreCves` contains individual ignored CVEs when determining policy enforcement. In the following example, an `app-operator` ignores `CVE-2018-14643` and `GHSA-f2jv-r9rf-7988` if they are false positives. See A Note on Vulnerability Scanners.

```
...
spec:
regoFile: |
  package main

  notAllowedSeverities := []
  ignoreCves := ["CVE-2018-14643", "GHSA-f2jv-r9rf-7988"]
...
```

2. Deploy the scan policy to the cluster:

```
kubectl apply -f <path_to_scan_policy>/<scan_policy_filename>.yaml -n <desired_
namespace>
```

For information about how scan policies are used in the CVE triage workflow, see Triaging and Remediating CVEs.

Further refine the Scan Policy for use

The scan policy earlier demonstrates how vulnerabilities are ignored during a compliance check. It is not possible to audit why a vulnerability is ignored. You might want to allow an exception, where a build with a failing vulnerability is allowed to progress through a supply chain. You can allow this exception for a certain period of time, requiring an expiration date. Vulnerability Exploitability Exchange (VEX) documents are gaining popularity to capture security advisory information pertaining to vulnerabilities. You can use Rego for these use cases.

For example, the following scan policy includes an additional text box to capture comments regarding why the scan ignores a vulnerability. The `notAllowedSeverities` array remains an array of strings, but the `ignoreCves` array updates from an array of strings to an array of objects. This causes a change to the `contains` function, splitting it into separate functions for each array.

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]

    # List of known vulnerabilities to ignore when deciding whether to fail complianc
e. Example:
```

```
    # ignoreCves := [
    #   {
    #     "id": "CVE-2018-14643",
    #     "detail": "Determined affected code is not in the execution path."
    #   }
    # ]
    ignoreCves := []

    containsSeverity(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := containsSeverity(notAllowedSeverities, severities[i])
      not fails
    }

    containsCve(array, elem) = true {
      array[_].id = elem
    } else = false { true }

    isSafe(match) {
      ignore := containsCve(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

The following example includes an expiration text box and only allows the vulnerability to be ignored for a period of time:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package main

    # Accepted Values: "Critical", "High", "Medium", "Low", "Negligible", "UnknownSeve
rity"
    notAllowedSeverities := ["Critical", "High", "UnknownSeverity"]

    # List of known vulnerabilities to ignore when deciding whether to fail complianc
e. Example:
    # ignoreCves := [
```

```
    #   {
    #     "id": "CVE-2018-14643",
    #     "detail": "Determined affected code is not in the execution path.",
    #     "expiration": "2022-Dec-31"
    #   }
    # ]
    ignoreCves := []

    containsSeverity(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isSafe(match) {
      severities := { e | e := match.ratings.rating.severity } | { e | e := match.rati
ngs.rating[_].severity }
      some i
      fails := containsSeverity(notAllowedSeverities, severities[i])
      not fails
    }

    containsCve(array, elem) = true {
      array[_].id = elem
      curr_time := time.now_ns()
      date_format := "2006-Jan-02"
      expire_time := time.parse_ns(date_format, array[_].expiration)
      curr_time < expire_time
    } else = false { true }

    isSafe(match) {
      ignore := containsCve(ignoreCves, match.id)
      ignore
    }

    deny[msg] {
      comps := { e | e := input.bom.components.component } | { e | e := input.bom.comp
onents.component[_] }
      some i
      comp := comps[i]
      vulns := { e | e := comp.vulnerabilities.vulnerability } | { e | e := comp.vulne
rabilities.vulnerability[_] }
      some j
      vuln := vulns[j]
      ratings := { e | e := vuln.ratings.rating.severity } | { e | e := vuln.ratings.r
ating[_].severity }
      not isSafe(vuln)
      msg = sprintf("CVE %s %s %s", [comp.name, vuln.id, ratings])
    }
```

Troubleshooting Rego files (Scan Policy)

To troubleshoot or confirm that any modifications made to the rego file in the provided sample scan policy are functioning as intended, see Troubleshooting Rego Files.

Enable Tanzu Application Platform GUI to view ScanPolicy Resource

For the Tanzu Application Platform GUI to view the ScanPolicy resource, it must have a matching `kubernetes-label-selector` with a `part-of` prefix.

The following example is portion of a ScanPolicy that is viewable by the Tanzu Application Platform GUI:

```
---
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanPolicy
metadata:
  name: scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    ...
```

Note

Anything van be a value for the label. The Tanzu Application Platform GUI is looking for the existence of the `part-of` prefix string and doesn't match for anything else specific.

Deprecated Rego file Definition

Before Scan Controller v1.2.0, you must use the following format where the rego file differences are:

- The package name must be `package policies` instead of `package main`.
- The deny rule is a Boolean `isCompliant` instead of `deny[msg]`.
    - **isCompliant rule:** The Rego file must define inside its body an `isCompliant` rule. This must be a Boolean type containing the result whether the vulnerability violates the security policy or not. If `isCompliant` is `true`, the vulnerability is allowed in the Source or Image scan. Otherwise, `false` is considered. Any scan that finds at least one vulnerability that evaluates to `isCompliant=false` makes the `PolicySucceeded` condition set to false.

The following is an example scan policy resource:

```
apiVersion: scanning.apps.tanzu.vmware.com/v1alpha1
kind: ScanPolicy
metadata:
  name: v1alpha1-scan-policy
  labels:
    app.kubernetes.io/part-of: enable-in-gui
spec:
  regoFile: |
    package policies

    default isCompliant = false

    ignoreSeverities := ["Critical", "High"]

    contains(array, elem) = true {
      array[_] = elem
    } else = false { true }

    isCompliant {
      ignore := contains(ignoreSeverities, input.currentVulnerability.Ratings.Rating
[_].Severity)
      ignore
    }
```

Create a ScanTemplate with Supply Chain Security Tools - Scan

This topic describes how to create a ScanTemplate with Supply Chain Security Tools - Scan.

Overview

The `ScanTemplate` custom resource (CR) defines how the scan Pod fulfills the task of vulnerability scanning. There are default `ScanTemplates` provided out of the box using the Tanzu Application Platform default scanner, `Anchore Grype`. One or more `initContainers` run to complete the scan and must save results to a shared `volume`. After the `initContainers` completes, a single container on the scan Pod called `summary` combines the result of the initContainers so that the `Scan CR` status is updated.

A customized ScanTemplate is created by editing or replacing `initContainer` definitions and reusing the `summary` container from the `grype` package. A container can read the `out.yaml` from an earlier step to locate relevant inputs.

Output Model

Each initContainer can create a subdirectory in `/workspace` to use as a scratch space. Before terminating the container must create an `out.yaml` file in the subdirectory containing the relevant subset of fields from the output model:

```
fetch:
  git:
    url:
    revision:
    path:
  blob:
    url:
    revision:
    path:
  image:
    url:
    revision:
    path:
sbom:
    packageCount:
    reports: []
scan:
  cveCount:
    critical:
    high:
    medium:
    low:
    unknown:
  scanner:
    name:
    vendor:
    version:
    db:
      version:
  reports: []
eval:
  violations: []
store:
  locations: []
```

The `scan` portion of the earlier output is required and if missing the scan controller fails to properly update the final status of the `Scan CR`. Other portions of the output, including those of `store` and `policy evaluation`, are optional and can be omitted if not applicable in a custom supply chain setup.

ScanTemplate Structure

```
apiVersion: scanning.apps.tanzu.vmware.com/v1beta1
kind: ScanTemplate
spec:
    template: # a core/v1 PodSpec
      # Here are list volumes mounted for writing to or
      # reading from during different stages of the scan
      volumes:
        # required the results of different scan stages
        # should be saved in files digestible by the scan
        # controller in this volume
        - name: workspace
        emptyDir: { }
      # different steps required for a scanning can be staged
      # in sequential stages through initContainers.
      initContainers:
      # Summary container will take results of initContainers
      # and will let Controller to update Scan CR status.
      containers:
        - name: summary
```

Note: You cannot name a container `sleep` because there is already a container named `sleep` which comes from the scan-link controller.

Sample Outputs

```
# example for a typical git clone (source scan fetch stage)
# saved at: /workspace/git-clone/out.yaml
fetch:
  git:
    url: github.com/my/repo
    revision: aee9f8
    path: /workspace/git-clone/cloned-repository
```

```
# an example of typical scan stage
# saved at: /workspace/grype-scan/out.yaml
scan:
  cveCount:
    critical: 0
    high: 1
    medium: 3
    low: 25
    unknown: 0
  scanner:
    name: grype
    vendor: Anchore
    version: 0.33.0
    db:
      version: 2022-04-13
  reports:
  - /workspace/grype-scan/repo.cyclonedx.xml
  - /workspace/grype-scan/app.cyclonedx.xml
  - /workspace/grype-scan/base.cyclonedx.xml
```

```
# example of a typical evaluation stage
# saved at: /workspace/policy-eval/out.yaml
eval:
  violations:
    - banned package log4j
    - critical CVE 2022-01-01-3333
    - number of critical CVEs over threshold
```

```
# example of a typical upload to store stage
# saved at: /workspace/upload-to-store/out.yaml
store:
  locations:
    - http://metadata-store.cluster.local:8080/reports/3
```

View scan status conditions for Supply Chain Security Tools - Scan

This topic explains how you can view scan status conditions for Supply Chain Security Tools - Scan.

Viewing scan status

You can view the scan status by using `kubectl describe` on a `SourceScan` or `ImageScan`. You can see information about the scan status under the Status field for each scan CR.

Overview of conditions

The `Status.Conditions` array is populated with the scan status information during and after scanning execution, and the policy validation (if defined for the scan) after the results are available.

Condition types for the scans

Scanning

The Condition with type `Scanning` indicates running the scanning TaskRun. The Status field indicates whether the scan is running or has already finished. For example, if `Status: True`, the scan TaskRun is still running and if `Status: False`, the scan is done.

The Reason field is `JobStarted` while the scan is running and `JobFinished` when it is done.

The Message fieldcan either be `The scan job is running` or `The scan job terminated` depending on the current Status and Reason.

Succeeded

The Condition with type `Succeeded` indicates the scanning TaskRun result. The Status field indicates whether the scan finished successfully or if it encountered an error. For example, the status is `Status: True` if it completed successfully or `Status: False` otherwise.

The Reason field is `JobFinished` if the scanning was successful or `Error` if otherwise.

The Message and Error fields have more information about the last seen status of the scan TaskRun.

SendingResults

The condition with type `SendingResults` indicates sending the scan results to the metadata store. In addition to a successful process of sending the results, the condition can also indicate that the

metadata store integration has not been configured or that there was an error sending. An error is usually a misconfigured metadata store URL or that the metadata store is inaccessible. Verify the installation steps to ensure that the configuration is correct regarding secrets being set within the `scan-link-system` namespace.

PolicySucceeded

The Condition with type `PolicySucceeded` indicates the compliance of the scanning results against the defined policies. See Code Compliance Policy Enforcement using Open Policy Agent (OPA). The Status field indicates whether the results are compliant or not (`Status: True` or `Status: False` respectively) or `Status: Unknown` in case an error occurred during the policy verification.

The Reason field is `EvaluationPassed` if the scan complies with the defined policies. The Reason field is `EvaluationFailed` if the scan is not compliant, or `Error` if something went wrong.

The Message and Error fields are populated with `An error has occurred` and an error message if something went wrong during policy verification. Otherwise, the Message field displays `No CVEs were found that violated the policy` if there are no non-compliant vulnerabilities found or `Policy violated because of X CVEs` indicating the count of unique vulnerabilities found.

Overview of CVECount

The `status.CVECount` is populated with the number of CVEs in each category (CRITICAL, HIGH, MEDIUM, LOW, UNKNOWN) and the total (CVETOTAL).

Note

You can also view scan CVE summary in print columns with `kubectl get` on a `SourceScan` or `ImageScan`.

Overview of MetadataURL

The `status.metadataURL` is populated with the URL of the vulnerability scan results in> the metadata store integration. This is only available when the integration is configured.

Overview of Phase

The `status.phase` field is populated with the current phase of the scan. The phases are: Pending, Scanning, Completed, Failed, and Error.

- `Pending`: initial phase of the scan.
- `Scanning`: execution of the scan TaskRun is running.
- `Completed`: scan completed and no CVEs were found that violated the scan policy.
- `Failed`: scan completed but CVEs were found that violated the scan policy.
- `Error`: indication of an error (e.g., an invalid scantemplate or scan policy).

Note

The PHASE print column also shows this with `kubectl get` on a `SourceScan` or `ImageScan`.

Overview of ScannedBy

The `status.scannedBy` field is populated with the name, vendor, and scanner version that generates the security assessment report.

Overview of ScannedAt

The `status.scannedAt` field is populated with the latest date when the scanning finishes.

Troubleshoot Rego files with a scan policy for Supply Chain Security Tools - Scan

This topic describes how you can use an example output to troubleshoot your Rego file for SCST - Scan. You use a Rego file in a scan policy custom resource. See Enforce compliance policy using Open Policy Agent.

For information about how to write Rego, see Open Policy Agent documentation.

Using the Rego playground

Use the Rego Playground, to evaluate your Rego file against an input. In this example, use the example output of an image or source scan custom resource.

Sample input in CycloneDX's XML re-encoded as JSON format

The following is an example scan custom resource output in CycloneDX's XML structure re-encoded as JSON. This example output contains CVEs at low, medium, high, and critical severities.

To troubleshoot using this example output:

1. Paste your Rego file and the example output into the Rego Playground.
2. Evaluate your Rego file against the example output and verify that your Rego file detects the intended CVEs. See this Rego example.

```
{
    "bom": {
        "-serialNumber": "urn:uuid:123",
        "-v": "http://cyclonedx.org/schema/ext/vulnerability/1.0",
        "-version": "1",
        "-xmlns": "http://cyclonedx.org/schema/bom/1.2",
        "components": {
            "component": [
                {
                    "-type": "library",
                    "licenses": {
                        "license": {
                            "name": "GPL-2"
                        }
                    },
                    "name": "adduser",
                    "version": "3.118",
                    "vulnerabilities": {
                        "vulnerability": [
                            {
                                "-ref": "urn:uuid:3d7c61c6-9cfa-494c-858a-9668a318ff2
3",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2011-3374"
                                },
```

```
                                "id": "CVE-2011-3374-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Low"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2011-3374"
                                }
                            },
                            {
                                "-ref": "urn:uuid:ebabaa92-2bf9-4d33-8181-595b0fdf55b
d",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-27350"
                                },
                                "id": "CVE-2020-27350-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Medium"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-27350"
                                }
                            },
                            {
                                "-ref": "urn:uuid:07c58c81-1e01-459d-9e9d-0e10456a9bf
0",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-3810"
                                },
                                "id": "CVE-2020-3810-a",
                                "ratings": {
                                    "rating": {
                                        "severity": "Medium"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-3810"
                                }
                            }
                        ]
                    }
                },
                {
                    "-type": "library",
                    "licenses": {
                        "license": [
                            {
                                "name": "GPL-2"
                            },
                            {
                                "name": "GPLv2+"
                            }
                        ]
                    },
                    "name": "apt",
```

```
                    "version": "1.8.2",
                    "vulnerabilities": {
                        "vulnerability": [
                            {
                                "-ref": "urn:uuid:3d7c61c6-9cfa-494c-858a-9668a318ff2
3",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2011-3374"
                                },
                                "id": "CVE-2011-3374",
                                "ratings": {
                                    "rating": {
                                        "severity": "Low"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2011-3374"
                                }
                            },
                            {
                                "-ref": "urn:uuid:ebabaa92-2bf9-4d33-8181-595b0fdf55b
d",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-27350"
                                },
                                "id": "CVE-2020-27350",
                                "ratings": {
                                    "rating": {
                                        "severity": "High"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-27350"
                                }
                            },
                            {
                                "-ref": "urn:uuid:07c58c81-1e01-459d-9e9d-0e10456a9bf
0",
                                "advisories": {
                                    "advisory": "https://security-tracker.debian.org/t
racker/CVE-2020-3810"
                                },
                                "id": "CVE-2020-3810",
                                "ratings": {
                                    "rating": {
                                        "severity": "Critical"
                                    }
                                },
                                "source": {
                                    "-name": "debian:10",
                                    "url": "http://cve.mitre.org/cgi-bin/cvename.cgi?n
ame=CVE-2020-3810"
                                }
                            }
                        ]
                    }
                }
            ]
        },
        "metadata": {
```

```
            "component": {
                "-type": "container",
                "name": "nginx:1.16",
                "version": "sha256:123"
            },
            "timestamp": "2022-01-28T13:30:43-08:00",
            "tools": {
                "tool": {
                    "name": "grype",
                    "vendor": "anchore",
                    "version": "[not provided]"
                }
            }
        }
    }
}
```

Example input in SPDX JSON format

The example in this section is a modified scan custom resource input, in `.spdx.json`, that contains CVEs at low, medium, high, and critical severities. You can use this example input to evaluate your Rego file.

To troubleshoot using this example output:

1. Paste your Rego file and the example input into the Rego Playground.
2. Evaluate your Rego file against the output and verify that your Rego file detects the intended CVEs. See this Rego example.

```
{
  "id": "SPDXRef-docker-image|nginx",
  "specVersion": "SPDX-3.0",
  "creator": "Organization: Snyk Ltd",
  "created": "2023-03-01T16:10:08Z",
  "profile": [
    "base",
    "vulnerabilities"
  ],
  "description": "Snyk test result for project docker-image|nginx in SPDX SBOM forma
t",
  "vulnerabilities": [
    {
      "id": "SNYK-DEBIAN10-APT-1049974",
      "name": "SNYK-DEBIAN10-APT-1049974",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nAPT had several integer overflows and underflows while parsing .deb packages, a
ka GHSL-2020-168 GHSL-2020-169, in files apt-pkg/contrib/extracttar.cc, apt-pkg/deb/de
bfile.cc, and apt-pkg/contrib/arfile.cc. This issue affects: apt 1.2.32ubuntu0 version
s prior to 1.2.32ubuntu0.2; 1.6.12ubuntu0 versions prior to 1.6.12ubuntu0.2; 2.0.2ubun
tu0 versions prior to 2.0.2ubuntu0.2; 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.
1;\n## Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.2 or higher.\n## Refere
nces\n- [ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-27350)\n- [CON
FIRM](https://bugs.launchpad.net/bugs/1899193)\n- [CONFIRM](https://security.netapp.co
m/advisory/ntap-20210108-0005/)\n- [DEBIAN](https://www.debian.org/security/2020/dsa-4
808)\n- [UBUNTU](https://usn.ubuntu.com/usn/usn-4667-1)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
```

```
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.7
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:L/I:L/A:L"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-27350"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1899193"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20210108-0005/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2020/dsa-4808"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/usn/usn-4667-1"
        }
      ],
      "modified": "2022-10-29T13:11:02.438923Z",
      "published": "2020-12-10T03:10:23.901831Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-407502",
      "name": "SNYK-DEBIAN10-APT-407502",
```

```
      "summary": "Improper Verification of Cryptographic Signature",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nIt was found that apt-key in apt, all versions, do not correctly validate gpg k
eys with the primary keyring, leading to a potential man-in-the-middle attack.\n## Rem
ediation\nThere is no fixed version for `Debian:10` `apt`.\n## References\n- [ADVISOR
Y](https://security-tracker.debian.org/tracker/CVE-2011-3374)\n- [Debian Bug Report](h
ttps://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480)\n- [MISC](https://people.cano
nical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html)\n- [MISC](https://seclists.or
g/fulldisclosure/2011/Sep/221)\n- [MISC](https://snyk.io/vuln/SNYK-LINUX-APT-116518)\n
- [MISC](https://ubuntu.com/security/CVE-2011-3374)\n- [RedHat CVE Database](https://a
ccess.redhat.com/security/cve/cve-2011-3374)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              347
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 3.7
                  }
                ],
                "severity": "Low",
                "vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480"
        },
```

```
        {
          "category": "ADVISORY",
          "locator": "https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-
3374.html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://seclists.org/fulldisclosure/2011/Sep/221"
        },
        {
          "category": "ADVISORY",
          "locator": "https://snyk.io/vuln/SNYK-LINUX-APT-116518"
        },
        {
          "category": "ADVISORY",
          "locator": "https://ubuntu.com/security/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://access.redhat.com/security/cve/cve-2011-3374"
        }
      ],
      "modified": "2022-11-01T00:08:27.375895Z",
      "published": "2018-06-27T16:20:45.037549Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-568926",
      "name": "SNYK-DEBIAN10-APT-568926",
      "summary": "Improper Input Validation",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nMissing input validation in the ar/tar implementations of APT before version 2.
1.2 could result in denial of service when processing specially crafted deb files.\n##
Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.1 or higher.\n## References\n-
[ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-3810)\n- [FEDORA](http
s://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/mes
sage/U4PEH357MZM2SUGKETMEHMSGQS652QHH/)\n- [GitHub Issue](https://github.com/Debian/ap
t/issues/111)\n- [MISC](https://bugs.launchpad.net/bugs/1878177)\n- [MISC](https://lis
ts.debian.org/debian-security-announce/2020/msg00089.html)\n- [MISC](https://salsa.deb
ian.org/apt-team/apt/-/commit/dceb1e49e4b8e4dadaf056be34088b415939cda6)\n- [MISC](http
s://tracker.debian.org/news/1144109/accepted-apt-212-source-into-unstable/)\n- [UBUNT
U](https://usn.ubuntu.com/4359-2/)\n- [Ubuntu CVE Tracker](http://people.ubuntu.com/~u
buntu-security/cve/CVE-2020-3810)\n- [Ubuntu Security Advisory](https://usn.ubuntu.co
m/4359-1/)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
```

```
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              20
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.5
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:H"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-3810"
        },
        {
          "category": "ADVISORY",
          "locator": "https://lists.fedoraproject.org/archives/list/package-announce@l
ists.fedoraproject.org/message/U4PEH357MZM2SUGKETMEHMSGQS652QHH/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/Debian/apt/issues/111"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1878177"
        },
        {
          "category": "ADVISORY",
          "locator": "https://lists.debian.org/debian-security-announce/2020/msg00089.
html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://salsa.debian.org/apt-team/apt/-/commit/dceb1e49e4b8e4dad
af056be34088b415939cda6"
        },
        {
          "category": "ADVISORY",
          "locator": "https://tracker.debian.org/news/1144109/accepted-apt-212-source-
into-unstable/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/4359-2/"
        },
        {
          "category": "ADVISORY",
          "locator": "http://people.ubuntu.com/~ubuntu-security/cve/CVE-2020-3810"
        },
        {
          "category": "ADVISORY",
```

```
          "locator": "https://usn.ubuntu.com/4359-1/"
        }
      ],
      "modified": "2022-11-01T00:08:51.907776Z",
      "published": "2020-05-12T14:19:01.052295Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-1049974",
      "name": "SNYK-DEBIAN10-APT-1049974",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nAPT had several integer overflows and underflows while parsing .deb packages, a
ka GHSL-2020-168 GHSL-2020-169, in files apt-pkg/contrib/extracttar.cc, apt-pkg/deb/de
bfile.cc, and apt-pkg/contrib/arfile.cc. This issue affects: apt 1.2.32ubuntu0 version
s prior to 1.2.32ubuntu0.2; 1.6.12ubuntu0 versions prior to 1.6.12ubuntu0.2; 2.0.2ubun
tu0 versions prior to 2.0.2ubuntu0.2; 2.1.10ubuntu0 versions prior to 2.1.10ubuntu0.
1;\n## Remediation\nUpgrade `Debian:10` `apt` to version 1.8.2.2 or higher.\n## Refere
nces\n- [ADVISORY](https://security-tracker.debian.org/tracker/CVE-2020-27350)\n- [CON
FIRM](https://bugs.launchpad.net/bugs/1899193)\n- [CONFIRM](https://security.netapp.co
m/advisory/ntap-20210108-0005/)\n- [DEBIAN](https://www.debian.org/security/2020/dsa-4
808)\n- [UBUNTU](https://usn.ubuntu.com/usn/usn-4667-1)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt@1.8.2",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 5.7
                  }
                ],
                "severity": "Medium",
                "vector": "CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:C/C:L/I:L/A:L"
              }
            ]
          }
```

```
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2020-27350"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.launchpad.net/bugs/1899193"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20210108-0005/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2020/dsa-4808"
        },
        {
          "category": "ADVISORY",
          "locator": "https://usn.ubuntu.com/usn/usn-4667-1"
        }
      ],
      "modified": "2022-10-29T13:11:02.438923Z",
      "published": "2020-12-10T03:10:23.901831Z"
    },
    {
      "id": "SNYK-DEBIAN10-APT-407502",
      "name": "SNYK-DEBIAN10-APT-407502",
      "summary": "Improper Verification of Cryptographic Signature",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `apt` package and not the `apt` package as distributed b
y `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and statu
s._\n\nIt was found that apt-key in apt, all versions, do not correctly validate gpg k
eys with the primary keyring, leading to a potential man-in-the-middle attack.\n## Rem
ediation\nThere is no fixed version for `Debian:10` `apt`.\n## References\n- [ADVISOR
Y](https://security-tracker.debian.org/tracker/CVE-2011-3374)\n- [Debian Bug Report](h
ttps://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480)\n- [MISC](https://people.cano
nical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html)\n- [MISC](https://seclists.or
g/fulldisclosure/2011/Sep/221)\n- [MISC](https://snyk.io/vuln/SNYK-LINUX-APT-116518)\n
- [MISC](https://ubuntu.com/security/CVE-2011-3374)\n- [RedHat CVE Database](https://a
ccess.redhat.com/security/cve/cve-2011-3374)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "apt@1.8.2",
              "apt/libapt-pkg5.0@1.8.2"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
```

```
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              347
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 3.7
                  }
                ],
                "severity": "Low",
                "vector": "CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480"
        },
        {
          "category": "ADVISORY",
          "locator": "https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-
3374.html"
        },
        {
          "category": "ADVISORY",
          "locator": "https://seclists.org/fulldisclosure/2011/Sep/221"
        },
        {
          "category": "ADVISORY",
          "locator": "https://snyk.io/vuln/SNYK-LINUX-APT-116518"
        },
        {
          "category": "ADVISORY",
          "locator": "https://ubuntu.com/security/CVE-2011-3374"
        },
        {
          "category": "ADVISORY",
          "locator": "https://access.redhat.com/security/cve/cve-2011-3374"
        }
      ],
      "modified": "2022-11-01T00:08:27.375895Z",
      "published": "2018-06-27T16:20:45.037549Z"
    },
    {
      "id": "SNYK-DEBIAN10-EXPAT-2329087",
      "name": "SNYK-DEBIAN10-EXPAT-2329087",
      "summary": "Incorrect Calculation",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `expat` package and not the `expat` package as distribut
ed by `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and st
atus._\n\nIn Expat (aka libexpat) before 2.4.3, a left shift by 29 (or more) places in
the storeAtts function in xmlparse.c can lead to realloc misbehavior (e.g., allocating
too few bytes, or only freeing memory).\n## Remediation\nUpgrade `Debian:10` `expat` t
```

```
o version 2.2.6-2+deb10u2 or higher.\n## References\n- [ADVISORY](https://security-tra
cker.debian.org/tracker/CVE-2021-45960)\n- [MISC](https://bugzilla.mozilla.org/show_bu
g.cgi?id=1217609)\n- [MISC](https://github.com/libexpat/libexpat/issues/531)\n- [MISC]
(https://github.com/libexpat/libexpat/pull/534)\n- [cve@mitre.org](http://www.openwal
l.com/lists/oss-security/2022/01/17/3)\n- [cve@mitre.org](https://security.netapp.com/
advisory/ntap-20220121-0004/)\n- [cve@mitre.org](https://www.tenable.com/security/tns-
2022-05)\n- [cve@mitre.org](https://www.debian.org/security/2022/dsa-5073)\n- [cve@mit
re.org](https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf)\n- [cve@mitre.
org](https://security.gentoo.org/glsa/202209-24)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "nginx-module-image-filter@1.16.1-1~buster",
              "libgd2/libgd3@2.2.5-5.2",
              "fontconfig/libfontconfig1@2.13.1-2",
              "expat/libexpat1@2.2.6-2+deb10u1"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              682
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 8.8
                  }
                ],
                "severity": "High",
                "vector": "CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2021-45960"
        },
        {
          "category": "ADVISORY",
          "locator": "https://bugzilla.mozilla.org/show_bug.cgi?id=1217609"
        },
        {
```

```
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/issues/531"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/pull/534"
        },
        {
          "category": "ADVISORY",
          "locator": "http://www.openwall.com/lists/oss-security/2022/01/17/3"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.netapp.com/advisory/ntap-20220121-0004/"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.tenable.com/security/tns-2022-05"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2022/dsa-5073"
        },
        {
          "category": "ADVISORY",
          "locator": "https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.gentoo.org/glsa/202209-24"
        }
      ],
      "modified": "2023-02-14T13:37:37.505975Z",
      "published": "2022-01-02T01:41:26.770663Z"
    },
    {
      "id": "SNYK-DEBIAN10-EXPAT-2331803",
      "name": "SNYK-DEBIAN10-EXPAT-2331803",
      "summary": "Integer Overflow or Wraparound",
      "details": "## NVD Description\n**_Note:_** _Versions mentioned in the descripti
on apply only to the upstream `expat` package and not the `expat` package as distribut
ed by `Debian:10`._\n_See `How to fix?` for `Debian:10` relevant fixed versions and st
atus._\n\ndefineAttribute in xmlparse.c in Expat (aka libexpat) before 2.4.3 has an in
teger overflow.\n## Remediation\nUpgrade `Debian:10` `expat` to version 2.2.6-2+deb10u
2 or higher.\n## References\n- [ADVISORY](https://security-tracker.debian.org/tracker/
CVE-2022-22824)\n- [cve@mitre.org](https://github.com/libexpat/libexpat/pull/539)\n-
[cve@mitre.org](http://www.openwall.com/lists/oss-security/2022/01/17/3)\n- [cve@mitr
e.org](https://www.tenable.com/security/tns-2022-05)\n- [cve@mitre.org](https://www.de
bian.org/security/2022/dsa-5073)\n- [cve@mitre.org](https://cert-portal.siemens.com/pr
oductcert/pdf/ssa-484086.pdf)\n- [cve@mitre.org](https://security.gentoo.org/glsa/2022
09-24)\n",
      "relationships": [
        {
          "affect": {
            "to": [
              "docker-image|nginx@1.16",
              "nginx-module-image-filter@1.16.1-1~buster",
              "libgd2/libgd3@2.2.5-5.2",
              "fontconfig/libfontconfig1@2.13.1-2",
              "expat/libexpat1@2.2.6-2+deb10u1"
            ],
            "type": "AFFECTS"
          },
          "foundBy": {
            "to": [
              ""
```

```
            ],
            "type": "FOUND_BY"
          },
          "suppliedBy": {
            "to": [
              ""
            ],
            "type": "SUPPLIED_BY"
          },
          "ratedBy": {
            "to": [
              ""
            ],
            "type": "RATED_BY",
            "cwes": [
              190
            ],
            "rating": [
              {
                "method": "CVSS_3",
                "score": [
                  {
                    "base": 9.8
                  }
                ],
                "severity": "Critical",
                "vector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H"
              }
            ]
          }
        }
      ],
      "externalReferences": [
        {
          "category": "ADVISORY",
          "locator": "https://security-tracker.debian.org/tracker/CVE-2022-22824"
        },
        {
          "category": "ADVISORY",
          "locator": "https://github.com/libexpat/libexpat/pull/539"
        },
        {
          "category": "ADVISORY",
          "locator": "http://www.openwall.com/lists/oss-security/2022/01/17/3"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.tenable.com/security/tns-2022-05"
        },
        {
          "category": "ADVISORY",
          "locator": "https://www.debian.org/security/2022/dsa-5073"
        },
        {
          "category": "ADVISORY",
          "locator": "https://cert-portal.siemens.com/productcert/pdf/ssa-484086.pdf"
        },
        {
          "category": "ADVISORY",
          "locator": "https://security.gentoo.org/glsa/202209-24"
        }
      ],
      "modified": "2023-02-14T13:39:18.516672Z",
      "published": "2022-01-08T13:52:14.479733Z"
    }
  ],
```

```
  "name": "docker-image|nginx-sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5e
eb69fcaaf30dd9c",
  "dataLicense": "CC0-1.0",
  "documentNamespace": "spdx.org/spdxdocs/docker-image|nginx-feb02ce6-cd47-49c2-9a97-2
b4833b4a1f0",
  "relationships": [
    {
      "from": "SPDXRef-docker-image|nginx",
      "to": [
        "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4807e0de4
62caf2336f0b70b5eeb69fcaaf30dd9c"
      ],
      "type": "DESCRIBES"
    }
  ],
  "packages": [
    {
      "SPDXID": "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4
807e0de462caf2336f0b70b5eeb69fcaaf30dd9c",
      "versionInfo": "sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaa
f30dd9c",
      "id": "SPDXRef-index.docker.io/library/nginx-sha256:d20aa6d1cae56fd17cd458f4807e
0de462caf2336f0b70b5eeb69fcaaf30dd9c",
      "name": "index.docker.io/library/nginx",
      "checksums": [
        {
          "algorithm": "SHA256",
          "checksumValue": "d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaaf
30dd9c"
        }
      ]
    }
  ]
}
```


Supply Chain Security Tools - Scan 2.0 (alpha)

This topic describes how you can install and configure Supply Chain Security Tools - Scan 2.0.

Important

SCST - Scan 2.0 is in Alpha, which means that it is still in active development by VMware and might be subject to change at any point. Users might encounter unexpected behavior due to capability gaps. This is an opt-in component to gather early feedback from alpha testers and is not installed by default with any profile.

Overview

SCST - Scan 2.0 is responsible for providing the framework to scan applications for their security posture. Scanning container images for known Common Vulnerabilities and Exposures (CVEs) implements this framework. This framework simplifies integration for new plug-ins by allowing users to integrate new scan engines by minimizing the scope of the scan engine to only scan and push results to an OCI compliant registry.

During scanning:

- A `GrypeImageVulnerabilityScan` creates the child resource `ImageVulnerabilityScan`.
- The `ImageVulnerabilityScan` then creates a Tekton PipelineRun which instantiates a Pipeline. The Pipeline Spec specifies the tasks `workspace-setup-task`, `scan-task`, and

`publish-task` to set up the workspace and environment configuration, run a scan, and publish results to an OCI compliant registry.

- Each Task contains steps which execute commands to achieve the goal of the Task.
- The PipelineRun creates corresponding TaskRuns for every Task in the Pipeline and executes them.
- A Tekton Sidecar as a no-op sidecar triggers Tekton's injected sidecar cleanup.

Note

SCST - Scan 2.0 is in Alpha and supersedes the SCST - Scan component.

Features

SCST - Scan 2.0 includes the following features:

- Tekton is used as the orchestrator of the scan to align with overall Tanzu Application Platform use of Tekton for multi-step activities.
- New scans are defined as Custom Resource Definitions (CRDs) that represent specific scanners, such as GrypeImageVulnerabilityScan. Mapping logic turns the domain-specific specifications into a Tekton PipelineRun.
- CycloneDX-formatted scan results are pushed to an OCI registry for long-term storage.

Installing SCST - Scan 2.0 in a cluster

The following sections describe how to install SCST - Scan 2.0.

Prerequisites

SCST - Scan 2.0 requires the following prerequisites:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install the Tekton component. Tekton is in the Full and Build profiles of Tanzu Application Platform.

Configure properties

When you install SCST - Scan 2.0, you can configure the following optional properties:

Key	Default	Type	Description
caCertData	""	string	The custom certificates trusted by the scan's connections
docker.import	true	Boolean	Import `docker.pullSecret` from another namespace (requires secretgen-controller). Set to false if the secret is already present.
docker.pullSecret	registries-credentials	string	Name of a Docker pull secret in the deployment namespace to pull the scanner images
workspace.storageSize	100Mi	string	Size of the PersistentVolume that the Tekton pipelineruns uses
workspace.storageClass	""	string	Name of the storage class to use while creating the PersistentVolume claims used by tekton pipelineruns

Install

To install SCST - Scan 2.0:

1. List version information for the package by running:

   ```
   tanzu package available list app-scanning.apps.tanzu.vmware.com --namespace tap
   -install
   ```

   For example:

   ```
   $ tanzu package available list app-scanning.apps.tanzu.vmware.com --namespace t
   ap-install
   - Retrieving package versions for app-scanning.apps.tanzu.vmware.com...
     NAME                                VERSION              RELEASED-AT
     app-scanning.apps.tanzu.vmware.com  0.1.0-alpha          2023-03-01 20:00:0
   0 -0400 EDT
   ```

2. (Optional) Make changes to the default installation settings:

   Create an `app-scanning-values-file.yaml` file which contains any changes to the default installation settings.

   Retrieve the configurable settings and append the key-value pairs to be modified to the `app-scanning-values-file.yaml` file:

   ```
   tanzu package available get app-scanning.apps.tanzu.vmware.com/VERSION --values
   -schema --namespace tap-install
   ```

   Where `VERSION` is your package version number. For example, `0.1.0-alpha`.

   For example:

   ```
   tanzu package available get app-scanning.apps.tanzu.vmware.com/0.1.0-alpha --va
   lues-schema --namespace tap-install
   | Retrieving package details for app-scanning.apps.tanzu.vmware.com/0.1.0-alph
   a...

     KEY                     DEFAULT                 TYPE     DESCRIPTION
     docker.import           true                    boolean  Import `docker.pullS
   ecret` from another namespace (requires
                                                            secretgen-controlle
   r). Set to false if the secret will already be present.
     docker.pullSecret       registries-credentials  string   Name of a docker pul
   l secret in the deployment namespace to pull the scanner
                                                            images.
     workspace.storageSize   100Mi                   string   Size of the Persiste
   nt Volume to be used by the tekton pipelineruns
     workspace.storageClass                          string   Name of the storage
   class to use while creating the Persistent Volume Claims
                                                            used by tekton pipel
   ineruns
     caCertData                                      string   The custom certifica
   tes to be trusted by the scan's connections
   ```

3. Install the package by running:

   ```
   tanzu package install app-scanning-alpha --package-name app-scanning.apps.tanz
   u.vmware.com \
       --version VERSION \
       --namespace tap-install \
       --values-file app-scanning-values-file.yaml
   ```

   Where `VERSION` is your package version number. For example, `0.1.0-alpha`.

For example:

```
tanzu package install app-scanning-alpha --package-name app-scanning.apps.tanz
u.vmware.com \
    --version 0.1.0-alpha \
    --namespace tap-install \
    --values-file app-scanning-values-file.yaml

    Installing package 'app-scanning.apps.tanzu.vmware.com'
    Getting package metadata for 'app-scanning.apps.tanzu.vmware.com'
    Creating service account 'app-scanning-default-sa'
    Creating cluster admin role 'app-scanning-default-cluster-role'
    Creating cluster role binding 'app-scanning-default-cluster-rolebinding'
    Creating package resource
    Waiting for 'PackageInstall' reconciliation for 'app-scanning'
    'PackageInstall' resource install status: Reconciling
    'PackageInstall' resource install status: ReconcileSucceeded
```

Configure namespace

The following sections describe how to configure service accounts and registry credentials.

The following access is required:

- Read access to the registry containing the Tanzu Application Platform bundles. This is the registry from the Relocate images to a registry step or `registry.tanzu.vmware.com`.
- Read access to the registry containing the image to scan, if scanning a private image
- Write access to the registry to which results are published
- Create a secret `scanning-tap-component-read-creds` with read access to the registry containing the Tanzu Application Platform bundles. This pulls the SCST - Scan 2.0 images.

  Important

  If you followed the directions for Install Tanzu Application Platform, skip this step and use the `tap-registry` secret with your service account.

  ```
  read -s TAP_REGISTRY_PASSWORD
  kubectl create secret docker-registry scanning-tap-component-read-creds \
    --docker-username=TAP-REGISTRY-USERNAME \
    --docker-password=$TAP_REGISTRY_PASSWORD \
    --docker-server=TAP-REGISTRY-URL \
    -n DEV-NAMESPACE
  ```

  Where `DEV-NAMESPACE` is the developer namespace where scanning occurs.

- If you are scanning a private image, create a secret `scan-image-read-creds` with read access to the registry containing that image.

  Important

  If you followed the directions for Install Tanzu Application Platform, you can skip this step and use the `targetImagePullSecret` secret with your service account as referenced in your tap-values.yaml here.

  ```
  read -s REGISTRY_PASSWORD
  kubectl create secret docker-registry scan-image-read-creds \
  ```

```
  --docker-username=REGISTRY-USERNAME \
  --docker-password=$REGISTRY_PASSWORD \
  --docker-server=REGISTRY-URL \
  -n DEV-NAMESPACE
```

- Create a secret `write-creds` with write access to the registry for the scanner to upload the scan results to.

```
read -s WRITE_PASSWORD
kubectl create secret docker-registry write-creds \
  --docker-username=WRITE-USERNAME \
  --docker-password=$WRITE_PASSWORD \
  --docker-server=DESTINATION-REGISTRY-URL \
  -n DEV-NAMESPACE
```

- Create the service account `scanner` which enables SCST - Scan 2.0 to pull the image to scan. Attach the read secret created earlier under `imagePullSecrets` and the write secret under `secrets`.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: scanner
  namespace: DEV-NAMESPACE
imagePullSecrets:
- name: scanning-tap-component-read-creds
secrets:
- name: scan-image-read-creds
```

  Where:

  - `imagePullSecrets.name` is the name of the secret used by the component to pull the scan component image from the registry.
  - `secrets.name` is the name of the secret used by the component to pull the image to scan. This is required if the image you are scanning is private.

- Create the service account `publisher` which enables SCST - Scan 2.0 to push the scan results to a user specified registry.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: publisher
  namespace: DEV-NAMESPACE
imagePullSecrets:
- name: scanning-tap-component-read-creds
secrets:
- name: write-creds
```

  Where: - `imagePullSecrets.name` is the name of the secret used by the component to pull the scan component image from the registry. - `secrets.name` is the name of the secret used by the component to publish the scan results.

Scan an image

The following section describes how to scan an image with SCST - Scan 2.0.

Retrieving an image digest

SCST - Scan 2.0 custom resources require the digest form of the URL. For example, `nginx@sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2`.

Use the Docker documentation to pull and inspect an image digest:

```
docker pull nginx:latest
docker inspect --format='{{index .RepoDigests 0}}' nginx:latest
```

Alternatively, you can install krane to retrieve the digest without pulling the image:

```
krane digest nginx:latest
```

Integrating with the Out of the Box Supply Chain

Authoring a ClusterImageTemplate

To create a ClusterImageTemplate to which you can incorporate a scanner of your choice, follow steps in Authoring a ClusterImageTemplate.

Configuring the supply chain

The `ImageVulnerabilityScan` is available to integrate into the Out of the Box Supply Chain with Testing and Scanning via either a user created ClusterImageTemplate or the following packaged ClusterImageTemplates: - `image-vulnerability-scan-grype` - `image-vulnerability-scan-trivy`

1. Complete the steps for Install Out of the Box Supply Chain with Testing and Scanning for Supply Chain Choreographer or confirm installation.
2. View available ClusterImageTemplates by running:

   ```
   kubectl get clusterimagetemplates | grep grype
   ```

3. Update your `tap-values.yaml` file to specify the ClusterImageTemplate. For example:

   ```
   ootb_supply_chain_testing_scanning:
   image_scanner_template_name: image-vulnerability-scan-grype
   ```

4. Update the TAP installation by running:

   ```
   tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
   lues-file tap-values.yaml -n tap-install
   ```

   - Where `TAP-VERSION` is the version of Tanzu Application Platform installed.

5. Create a sample workload using a pre-built image by using the `tanzu apps workload create` command:

   ```
   tanzu apps workload create WORKLOAD-NAME \
   --app APP-NAME \
   --type TYPE \
   --image IMAGE \
   --namespace DEV-NAMESPACE
   ```

Where: - `WORKLOAD-NAME` is the name you choose for your workload. - `APP-NAME` is the name of your app. - `TYPE` is the type of your app. - `IMAGE` is the container image that contains the app you want to deploy. - `DEV-NAMESPACE` is the name of the developer namespace where scanning occurs.

Note: There are specific requirements for pre-built images. For more details see Configure your workload to use a prebuilt image

1. (Optional) Results will be pushed to the Artifactory Metadata Repository but if you wish to verify independently, you can use the following command to review the scan results.

   ```
   results=$(kubectl get imagevulnerabilityscan <IVS-NAME> -n DEV-NAMESPACE -o jso
   npath="{.status.scanResult}")

   imgpkg pull -b $results -o /tmp/scan-results
   ```

   Where:

   - `IVS-NAME` is the name of the ImageVulnerabilityScan.
   - `DEV-NAMESPACE` is the name of the developer namespace where scanning occurs.

Note: SCST - Scan 2.0 is in Beta and active keychains and workspace bindings are not modifiable in the packaged ClusterImageTemplates.

Using the provided Grype scanner

The following sections describe how to use Grype with SCST - Scan 2.0.

Sample Grype scan

To create a sample Grype scan:

1. Create a file named `grype-image-vulnerability-scan.yaml`. Configure the `image` and `scanResults.location`:

   ```
   apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
   kind: GrypeImageVulnerabilityScan
   metadata:
     name: grypescan
     namespace: DEV-NAMESPACE
   spec:
     image: nginx@sha256:... # The image to be scanned. Digest must be specified.
     scanResults:
       location: registry/project/scan-results # Registry to upload scan results
     serviceAccountNames:
       scanner: scanner # Service account that enables scanning component to pull
   the image to be scanned
       publisher: publisher # Service account has the secrets to push the scan res
   ults
   ```

Configuration Options

This section describes optional and required GrypeImageVulnerabilityScan specifications.

Required fields:

- `image` is the registry URL and digest of the scanned image. For example, `nginx@sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2`.
- `scanResults.location` is the registry URL where results are uploaded. For example, `my.registry/scan-results`.

Optional fields:

- `activeKeychains` is an array of enabled credential helpers to authenticate against registries using workload identity mechanisms. For more information, see the documentation for your cloud registry.

  ```
  activeKeychains:
  - name: acr  # Azure Container Registry
  ```

```
- name: ecr  # Elastic Container Registry
- name: gcr  # Google Container Registry
- name: ghcr # Github Container Registry
```

- `advanced` is the adjusted configuration of Grype for your needs. See the Grype documentation.
- `serviceAccountNames` includes:
  - `scanner` is the service account that runs the scan. It must have read access to `image`.
  - `publisher` is the service account that uploads results. It must have write access to `scanResults.location`.
- `workspace` includes:
  - `size` is the size of the PersistentVolumeClaim the scan uses to download the image and vulnerability database.
  - `bindings` are additional array of secrets, ConfigMaps, or EmptyDir volumes to mount to the running scan. The `name` is used as the mount path.

```
bindings:
- name: additionalconfig
  configMap:
    name: my-configmap
- name: additionalsecret
  secret:
    secretName: my-secret
- name: scratch
  emptyDir: {}
```

For information about workspace bindings, see Using other types of volume sources. Only Secrets, ConfigMaps, and EmptyDirs are supported.

Trigger a Grype scan

To trigger a Grype scan:

1. Apply the `GrypeImageVulnerabilityScan` to the cluster.

   ```
   kubectl apply -f grype-image-vulnerability-scan.yaml -n DEV-NAMESPACE
   ```

2. The Grype scan creates child resources.
   - View the child ImageVulnerabilityScan by running:

     ```
     kubectl get imagevulnerabilityscan -n DEV-NAMESPACE
     ```

   - View the child PipelineRun, TaskRuns, and pods by running:

     ```
     kubectl get -l imagevulnerabilityscan pipelinerun,taskrun,pod -n DEV-NAME
     SPACE
     ```

3. When the scanning completes, the status is shown. Use `-o wide` to see the digest of the image scanned and the location of the published results.

   ```
   kubectl get grypeimagevulnerabilityscans grypescan -n DEV-NAMESPACE -o wide

   NAME        SCANRESULT                          SCANNEDIMAGE          SUCCEEDE
   D   REASON
   grypescan   registry/project/scan-results@digest nginx:latest@digest   True
   ```

```
Succeeded
```

Integrate your own scanner

To scan with any other scanner, use the generic `ImageVulnerabilityScan`. ImageVulnerabilityScans can also change the version of a scanner or customize the behavior of provided scanners.

ImageVulnerabilityScans allow you to define your scan as a Tekton step

Sample ImageVulnerabilityScan

To create a sample Sample ImageVulnerabilityScan:

1. Create a file named `image-vulnerability-scan.yaml`. Configure the `image`, `scanResults.location` of the scan, and define the scanner `image`, `command`, and `args` for your scanner `step`:

   ```
   apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
   kind: ImageVulnerabilityScan
   metadata:
     name: generic-image-scan
     namespace: DEV-NAMESPACE
   spec:
     image: nginx@sha256:...
     scanResults:
       location: registry/project/scan-results
     serviceAccountNames:
       scanner: scanner
       publisher: publisher
     steps:
     - name: scan
       image: anchore/grype:latest
       command: ["grype"]
       args:
       - registry:$(params.image)
       - -o
       - cyclonedx
       - --file
       - $(params.scan-results-path)/scan.cdx
   ```

   Where `DEV-NAMESPACE` is the developer namespace where scanning occurs.

   Note: Do not define `write-certs` or `cred-helper` as step names. These names are already used in steps during scanning.

Configuration options

This section lists optional and required ImageVulnerabilityScan specifications fields.

Required fields:

- `image` is the registry URL and digest of the image to scan. For example, `nginx@sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2`.
- `scanResults.location` is the registry URL where results are uploaded. For example, `my.registry/scan-results`.

Optional fields:

- `activeKeychains` is an array of enabled credential helpers to authenticate against registries using workload identity mechansims. See cloud registry documentation for details.

```
activeKeychains:
- name: acr  # Azure Container Registry
- name: ecr  # Elastic Container Registry
- name: gcr  # Google Container Registry
- name: ghcr # Github Container Registry
```

- `serviceAccountNames` includes:
    - `scanner` is the service account that runs the scan. It must have read access to `image`.
    - `publisher` is the service account that uploads results. It must have write access to `scanResults.location`.
- `workspace` includes:
    - `size` is size of the PersistentVolumeClaim the scan uses to download the image and vulnerability database.
    - `bindings` are additional array of secrets, ConfigMaps, or EmptyDir volumes to mount to the running scan. The `name` is used as the mount path.

```
bindings:
- name: additionalconfig
  configMap:
    name: my-configmap
- name: additionalsecret
  secret:
    secretName: my-secret
- name: scratch
  emptyDir: {}
```

For information about workspace bindings, see Using other types of volume sources. Only Secrets, ConfigMaps, and EmptyDirs are supported.

Default environment

Tekton Workspaces:

- `/home/app-scanning`: a memory-backed EmptyDir mount that contains service account credentials loaded by Tekton
- `/cred-helper`: a memory-backed EmptyDir mount containing:
    - config.json which combines static credentials with workload identity credentials when `activeKeychains` is enabled
    - trusted-cas.crt when SCST - Scan 2.0 is deployed with `caCertData`
- `/workspace`: a PersistentVolumeClaim to hold scan artifacts and results
    - The working directory for all Steps is by default located at `/workspace/scan-results`

Environment Variables: If undefined by your `step` definition the environment uses the following default variables:

- HOME=/home/app-scanning
- DOCKER_CONFIG=/cred-helper
- XDG_CACHE_HOME=/workspace/.cache
- TMPDIR=/workspace/tmp
- SSL_CERT_DIR=/etc/ssl/certs:/cred-helper

Tekton Pipelines Parameters:

These parameters are populated after creating the GrypeImageVulnerabilityScan. For information about parameters, see the Tekton documentation.

Parameters	Default	Type	Description
image	""	string	The scanned image
scan-results-path	/workspace/scan-results	string	Location to save scanner output
trusted-ca-certs	""	string	PEM data from the installation's `caCertData`

Trigger your scan

To trigger your scan:

1. Deploy your ImageVulnerabilityScan to the cluster by running:

   ```
   kubectl apply -f image-vulnerability-scan.yaml -n DEV-NAMESPACE
   ```

2. Child resources are created.

   - view the child PipelineRun, TaskRuns, and pods

     ```
     kubectl get -l imagevulnerabilityscan pipelinerun,taskrun,pod -n DEV-NAME
     SPACE
     ```

3. When the scanning completes, the status is shown. Specify `-o wide` to see the digest of the image scanned and the location of the published results.

   ```
   $ kubectl get imagevulnerabilityscans -n DEV-NAMESPACE -o wide

   NAME                 SCANRESULT                            SCANNEDIMAGE
   SUCCEEDED   REASON
   generic-image-scan   registry/project/scan-results@digest nginx:latest@digest
   True        Succeeded
   ```

Retrieving results

Scan results are uploaded to the container image registry as an imgpkg bundle. To retrieve a vulnerability report:

1. Retrieve the result location from the ImageVulnerabilityScan CR Status

   ```
   SCAN_RESULT_URL=$(kubectl get imagevulnerabilityscan my-scan -o jsonpath='{.sta
   tus.scanResult}')
   ```

2. Download the bundle to a local directory and list the content

   ```
   imgpkg pull -b $SCAN_RESULT_URL -o myresults/
   ls myresults/
   ```

Observability

To watch the status of the scanning custom resources and child resources:

```
kubectl get grypeimagevulnerabilityscan,imagevulnerabilityscan
kubectl get -l imagevulnerabilityscan pipelinerun,taskrun,pod
```

View the status, reason, and urls:

```
kubectl get grypeimagevulnerabilityscan -o wide
kubectl get imagevulnerabilityscan -o wide
```

View the complete status and events of scanning custom resources:

```
kubectl describe grypeimagevulnerabilityscan
kubectl describe imagevulnerabilityscan
```

List the child resources of a scan:

```
kubectl get -l grypeimagevulnerabilityscan=$NAME pipelinerun,taskrun,pod,configmap
kubectl get -l imagevulnerabilityscan=$NAME pipelinerun,taskrun,pod
```

Get the logs of the controller:

```
kubectl logs -f deployment/app-scanning-controller-manager -n app-scanning-system -c m
anager
```

Troubleshooting

Debugging commands

The following sections describe commands you can run to get logs and details about scanning errors.

Debugging resources

If a resource fails or has errors, inspect the resource.

To get status conditions on a resource:

```
kubectl describe RESOURCE RESOURCE-NAME -n DEV-NAMESPACE
```

Where:

- `RESOURCE` is one of the following: `GrypeImageVulnerabilityScan`, `ImageVulnerabilityScan`, `PipelineRun`, or `TaskRun`.
- `RESOURCE-NAME` is the name of the `RESOURCE`.
- `DEV-NAMESPACE` is the name of the developer namespace you want to use.

Debugging scan pods

To get error logs from a pod when scan pods fail:

```
kubectl logs SCAN-POD-NAME -n DEV-NAMESPACE
```

Where `SCAN-POD-NAME` is the name of the scan pod.

For information about debugging Kubernetes pods, see the Kubernetes documentation.

A scan run that has an error means that one of the following step containers has a failure:

- `step-write-certs`
- `step-cred-helper`
- `step-publisher`
- `sidecar-sleep`

- `working-dir-initializer`

To determine which step container had a failed exit code:

```
kubectl get taskrun TASKRUN-NAME -o json | jq .status
```

Where `TASKRUN-NAME` is the name of the TaskRun.

To inspect a specific step container in a pod:

```
kubectl logs scan-pod-name -n DEV-NAMESPACE -c step-container-name
```

Where `DEV-NAMESPACE` is your developer namespace.

For information about debugging a TaskRun, see the Tekton documentation.

Viewing the Scan-Controller manager logs

To retrieve scan-controller manager logs:

```
kubectl logs deployment/app-scanning-controller-manager -n app-scanning-system
```

To tail scan-controller manager logs:

```
kubectl logs -f deployment/app-scanning-controller-manager -n app-scanning-system
```

Author a ClusterImageTemplate for Supply Chain integration

This topic tells you how to create your own ClusterImageTemplate and customize the embedded ImageVulnerabilityScan to use the scanner of your choice.

Create a ClusterImageTemplate

To create a ClusterImageTemplate using an ImageVulnerabilityScan with Trivy:

1. Create a file with the following content and name it `custom-ivs-template.yaml`

```
apiVersion: carto.run/v1alpha1
kind: ClusterImageTemplate
metadata:
name: image-vulnerability-scan-custom # input name of your ClusterImageTemplate
spec:
imagePath: .status.scannedImage
retentionPolicy:
  maxFailedRuns: 10
  maxSuccessfulRuns: 10
lifecycle: immutable

healthRule:
  multiMatch:
    healthy:
      matchConditions:
        - status: "True"
          type: ScanCompleted
        - status: "True"
          type: Succeeded
    unhealthy:
      matchConditions:
        - status: "False"
          type: ScanCompleted
```

```
        - status: "False"
          type: Succeeded

params:
  - name: image_scanning_workspace_size
    default: 3Gi
  - name: image_scanning_service_account_scanner
    default: scanner
  - name: image_scanning_service_account_publisher
    default: publisher
  - name: trivy_db_repository
    default: ghcr.io/aquasecurity/trivy-db
  - name: trivy_java_db_repository
    default: ghcr.io/aquasecurity/trivy-java-db
  - name: registry-server
    default: my-registry.io    # input your registry server
  - name: registry-repository
    default: my-registry-repository    # input your registry repository

ytt: |
  #@ load("@ytt:data", "data")

  #@ def merge_labels(fixed_values):
  #@   labels = {}
  #@   if hasattr(data.values.workload.metadata, "labels"):
  #@     labels.update(data.values.workload.metadata.labels)
  #@   end
  #@   labels.update(fixed_values)
  #@   return labels
  #@ end

  #@ def scanResultsLocation():
  #@   return "/".join([
  #@    data.values.params.registry-server,
  #@    data.values.params.registry-repository,
  #@    "-".join([
  #@      data.values.workload.metadata.name,
  #@      data.values.workload.metadata.namespace,
  #@      "scan-results",
  #@    ])
  #@   ]) + ":" + data.values.workload.metadata.uid
  #@ end

  ---
  apiVersion: app-scanning.apps.tanzu.vmware.com/v1alpha1
  kind: ImageVulnerabilityScan
  metadata:
    labels: #@ merge_labels({ "app.kubernetes.io/component": "image-scan" })
    generateName: #@ data.values.workload.metadata.name + "-trivy-scan-"
  spec:
    image: #@ data.values.image
    scanResults:
      location: #@ scanResultsLocation()
    workspace:
      size: #@ data.values.params.image_scanning_workspace_size
    serviceAccountNames:
      scanner: #@ data.values.params.image_scanning_service_account_scanner
      publisher: #@ data.values.params.image_scanning_service_account_publisher
    steps:
    - name: trivy-generate-report
      image: my.registry.com/aquasec/trivy:0.41.0     # input the location of y
our trivy scanner image
      env:
      - name: TRIVY_DB_REPOSITORY
        value: #@ data.values.params.trivy_db_repository
      - name: TRIVY_JAVA_DB_REPOSITORY
```

```
        value: #@ data.values.params.trivy_java_db_repository
      - name: TRIVY_CACHE_DIR
        value: /workspace/trivy-cache
      - name: XDG_CACHE_HOME
        value: /workspace/.cache
      - name: TMPDIR
        value: /workspace
      args:
      - image
      - $(params.image)
      - --exit-code=0
      - --no-progress
      - --scanners=vuln
      - --format=cyclonedx
      - --output=scan.cdx.json
    - name: trivy-display-report
      image: my.registry.com/aquasec/trivy:0.41.0     # input the location of y
our trivy scanner image
      env:
      - name: TRIVY_DB_REPOSITORY
        value: #@ data.values.params.trivy_db_repository
      - name: TRIVY_JAVA_DB_REPOSITORY
        value: #@ data.values.params.trivy_java_db_repository
      - name: TRIVY_CACHE_DIR
        value: /workspace/trivy-cache
      - name: XDG_CACHE_HOME
        value: /workspace/.cache
      - name: TMPDIR
        value: /workspace
      args:
      - image
      - $(params.image)
      - --skip-db-update
      - --skip-java-db-update
      - --exit-code=0
      - --scanners=vuln
      - --severity=HIGH
      - --no-progress
```

Where:

- `.metadata.name` is the name of your ClusterImageTemplate. It must not conflict with the names of packaged templates. See Author your supply chains.
- `registry-server` is the registry server.
- `registry-repository` is the registry repository.

2. Edit your `custom-ivs-template.yaml` to update the name of your ClusterImageTemplate, the registry fields for your registry, and the location of your Trivy scanner image.
3. Create the ClusterImageTemplate:

```
kubectl apply -f custom-ivs-template.yaml
```

4. After you created your custom ClusterImageTemplate, you can proceed to integrating it in the Supply Chain

Overview of Supply Chain Security Tools for VMware Tanzu - Sign

This component is removed in Tanzu Application Platform v1.4 in favor of Supply Chain Security Tools - Policy Controller.

To migrate from Supply Chain Security Tools - Sign to Supply Chain Security Tools - Policy Controller, see Migration From Supply Chain Security Tools - Sign

Overview of Supply Chain Security Tools for Tanzu – Store

This topic gives you an overview of Supply Chain Security Tools (SCST) – Store.

Overview

Supply Chain Security Tools - Store saves software bills of materials (SBoMs) to a database and allows you to query for image, source code, package, and vulnerability relationships. It integrates with Supply Chain Security Tools - Scan to automatically store the resulting source code and image vulnerability reports. It accepts CycloneDX input and outputs in both human-readable and machine-readable formats, including JSON, text, and CycloneDX.

The following is a quick demo of configuring the tanzu insight plug-in and querying the metadata store for CVEs and scan results.

Using the Tanzu Insight CLI plug-in

The Tanzu Insight CLI plug-in is the primary way to view results from the Supply Chain Security Tools - Scan of source code and image files. Use it to query by source code commit, image digest, and CVE identifier to understand security risks.

See Tanzu Insight plug-in overview to install, configure, and use `tanzu insight`.

Multicluster configuration

See Multicluster setup for information about how to set up SCST - Store in a multicluster setup.

Integrating with Tanzu Application Platform GUI

Using the Supply Chain Choreographer in Tanzu Application Platform GUI, you can visualize your supply chain. It uses SCST - Store to show the packages and vulnerabilities in your source code and images.

To enable this feature, see Supply Chain Choreographer in Tanzu Application Platform GUI - Enable CVE scan results.

Additional documentation

Additional documentation includes information about the API, deployment details and configuration, AWS RDS configuration, other database backup recommendations, known issues, and other topics.

Overview of Supply Chain Security Tools for Tanzu – Store

This topic gives you an overview of Supply Chain Security Tools (SCST) – Store.

Overview

Supply Chain Security Tools - Store saves software bills of materials (SBoMs) to a database and allows you to query for image, source code, package, and vulnerability relationships. It integrates with Supply Chain Security Tools - Scan to automatically store the resulting source code and image vulnerability reports. It accepts CycloneDX input and outputs in both human-readable and machine-readable formats, including JSON, text, and CycloneDX.

The following is a quick demo of configuring the tanzu insight plug-in and querying the metadata store for CVEs and scan results.

Using the Tanzu Insight CLI plug-in

The Tanzu Insight CLI plug-in is the primary way to view results from the Supply Chain Security Tools - Scan of source code and image files. Use it to query by source code commit, image digest, and CVE identifier to understand security risks.

See Tanzu Insight plug-in overview to install, configure, and use `tanzu insight`.

Multicluster configuration

See Multicluster setup for information about how to set up SCST - Store in a multicluster setup.

Integrating with Tanzu Application Platform GUI

Using the Supply Chain Choreographer in Tanzu Application Platform GUI, you can visualize your supply chain. It uses SCST - Store to show the packages and vulnerabilities in your source code and images.

To enable this feature, see Supply Chain Choreographer in Tanzu Application Platform GUI - Enable CVE scan results.

Additional documentation

Additional documentation includes information about the API, deployment details and configuration, AWS RDS configuration, other database backup recommendations, known issues, and other topics.

Configure your target endpoint and certificate for Supply Chain Security Tools - Store

This topic describes how you can configure your target endpoint and certificate for Supply Chain Security Tools (SCST) - Store.

Overview

The connection to Supply Chain Security Tools - Store requires TLS encryption, and the configuration depends on the kind of installation.

For a production environment, VMware recommends that SCST - Store is installed with ingress enabled. The following instructions help set up the TLS connection, assuming that you deployed with ingress enabled.

Using `Ingress`

When using an Ingress setup, SCST - Store creates a specific TLS Certificate for HTTPS communications under the `metadata-store` namespace.

Set the endpoint host to `metadata-store.INGRESS-DOMAIN`, such as `metadata-store.example.domain.com`. Where `INGRESS-DOMAIN` isthe value of the `ingress_domain` property in your deployment yaml.

Note In a multi-cluster setup, a DNS record is **required** for the domain. The below instructions for single cluster setup do not apply, skip to Set Target section.

Single Cluster setup

In a single-cluster setup, a DNS record is still recommended. However, if no accessible DNS record exists for the domain, edit the `/etc/hosts` file to add a local record:

```
ENVOY_IP=$(kubectl get svc envoy -n tanzu-system-ingress -o jsonpath="{.status.loadBal
ancer.ingress[0].ip}")

# Replace with your domain
METADATA_STORE_DOMAIN="metadata-store.example.domain.com"

# Delete any previously added entry
sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "$ENVOY_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

Set Target

To get the certificate, run:

```
kubectl get secret tap-ingress-selfsigned-root-ca -n cert-manager -o json | jq -r '.da
ta."ca.crt"' | base64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN --ca-cert insight-ca.cr
t
```

Important

The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to add data and query data.

Next Step

- Configure access token

Additional Resources

For information about deploying SCST - Store **without** Ingress, see:

- Using LoadBalancer
- Using NodePort

Configure your access tokens for Supply Chain Security Tools - Store

This topic describes how to configure your access tokens for Supply Chain Security Tools - Store.

The access token is a `Bearer` token used in the http request header `Authorization`. For example, `Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhMV0....`

Service accounts are required to have associated access tokens. Before Kubernetes 1.24, service accounts generated access tokens automatically. Since Kubernetes 1.24, a secret must be applied manually.

By default, Supply Chain Security Tools - Store includes a `read-write` service account installed with an access token generated. This service account is cluster-wide. If you want to create your own service accounts, see Create Service Accounts.

Setting the Access Token

When using the `insight` plug-in, you must set the `METADATA_STORE_ACCESS_TOKEN` environment variable, or use the `--access-token` flag. VMware discourages using the `--access-token` flag as the token appears in your shell history.

The following command retrieves the access token from the default `metadata-store-read-write-client` service account and stores it in `METADATA_STORE_ACCESS_TOKEN`:

```
export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets metadata-store-read-write-cli
ent -n metadata-store -o jsonpath="{.data.token}" | base64 -d)
```

Additional Resources

- Retrieve access tokens

- Create service accounts
- Create a service account with a custom cluster role

Security details for Supply Chain Security Tools - Store

This topic describes the security details for Supply Chain Security Tools (SCST) - Store.

Application security

TLS encryption

Supply Chain Security Tools - Store requires TLS connection. If certificates are not provided, the application does not start. It supports TLS v1.2 and TLS v1.3. It does not support TLS 1.0, so a downgrade attack cannot happen. TLS 1.0 is prohibited under Payment Card Industry Data Security Standard (PCI DSS).

Cryptographic algorithms

Elliptic Curve:

```
CurveP521
CurveP384
CurveP256
```

Cipher Suites:

```
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

Access controls

SCST - Store uses kube-rbac-proxy as the only entry point to its API. Authentication and Authorization must be completed by using the `kube-rbac-proxy` before its API is accessible.

Authentication

The `kube-rbac-proxy` uses Token Review to verify that the token is valid. `Token Review` is a Kubernetes API to ensure that a trusted vendor issued the access token provided by the user. To issue an access token using Kubernetes, the user can create a Kubernetes Service Account and retrieve the corresponding generated secret for the access token.

To create a service account and use its access token, see the Create Service Account Docs.

Authorization

The `kube-rbac-proxy` uses Subject Access Review to ensure that users access certain operations. `Subject Access Review` is a Kubernetes API that uses Kubernetes RBAC to verify that the user can perform specific actions. See Create Service Account Doc.

There are two supported roles:

- `Read Only` cluster role
- `Read and Write` cluster role

These cluster roles are deployed by default. Additionally, a service account is created and bound to the `Read and Write` cluster role by default. If you do not want this service account, set the `add_default_rw_service_account` property to `false` in the `metadata-store-values.yaml` file durring deployment. See Install SCST - Store.

There is no default service account bound to the `Read Only` cluster role. You must create your service account and cluster role binding to bind to the `Read Only` role.

Important

There is no support for roles with access to only specific types of resources For example, images, packages, and vulnerabilities.

Container security

Non-root user

All containers shipped do not use root user accounts or accounts with root access. Using Kubernetes Security Context ensures that applications do not run with root users.

Security Context for the API server:

```
allowPrivilegeEscalation: false
runAsUser: 65532
fsGroup: 65532
```

Security Context for the PostgreSQL database pod:

```
allowPrivilegeEscalation: false
runAsUser: 999
fsGroup: 999
```

Note

`65532` is the UUID for the nobody user. `999` is the UUID for the PostgreSQL user.

Security scanning

There are two types of security scans that are performed before every release.

Static Application Security Testing (SAST)

A Coverity Scan is run on the source code of the API server, CLI, and all their dependencies. There are no high or critical items outstanding at the time of release.

Software Composition Analysis (SCA)

A Black Duck scan is run on the compiled binary to check for vulnerabilities and license data. There are no high or critical items outstanding at the time of release.

A Grype scan is run against the source code and the compiled container for dependencies vulnerabilities. There are no high or critical items outstanding at the time of release.

Additional documentation for Supply Chain Security Tools - Store

This topic describes additional documentation you can use with Supply Chain Security Tools - Store.

Use and operate

- Multicluster setup
- Developer namespace setup
- API details
- API walkthrough
- Failover, redundancy, and backups

Troubleshooting and logging

- Troubleshooting upgrading
- Log configuration and usage
- Connecting to the Postgres Database

Configuration

- Deployment details and configuration

Access control

- Retrieve access tokens
- Create service accounts
- Create a service account with a custom cluster role

Certificates

- Ingress support
- Using LoadBalancer
- Using NodePort
- Custom certificate configuration
- TLS configuration
- Certificate rotation
- Scanners cluster specific configurations

Database

- Use external postgres database
- AWS RDS postgres configuration
- Database backup recommendations

Other

- Install SCST - Store independent from TAP profiles

Additional documentation for Supply Chain Security Tools - Store

This topic describes additional documentation you can use with Supply Chain Security Tools - Store.

Use and operate

- Multicluster setup
- Developer namespace setup
- API details
- API walkthrough
- Failover, redundancy, and backups

Troubleshooting and logging

- Troubleshooting upgrading
- Log configuration and usage
- Connecting to the Postgres Database

Configuration

- Deployment details and configuration

Access control

- Retrieve access tokens
- Create service accounts
- Create a service account with a custom cluster role

Certificates

- Ingress support
- Using LoadBalancer
- Using NodePort
- Custom certificate configuration
- TLS configuration
- Certificate rotation
- Scanners cluster specific configurations

Database

- Use external postgres database
- AWS RDS postgres configuration
- Database backup recommendations

Other

- Install SCST - Store independent from TAP profiles

API reference for Supply Chain Security Tools - Store

This topic contains API reference information for Supply Chain Security Tools - Store. See API walkthrough for an SCST - Store example.

Information

Version

1.4.1

Content negotiation

URI Schemes

- http
- https

Consumes

- application/json

Produces

- application/json

All endpoints

images

Method	URI	Name	Summary
POST	/api/imageReport	create image report	Create a new image report. Related packages and vulnerabilities are also created.
GET	/api/images	get images	Search image by id, name or digest .
GET	/api/packages/{IDorName}/images	get package images	List the images that contain the given package.
GET	/api/vulnerabilities/{CVEID}/images	get vulnerability images	List the images that contain the given vulnerability.

Operations

Method	URI	Name	Summary
GET	/api/health	health check	

Packages

Method	URI	Name	Summary
GET	/api/images/{IDorDigest}/packages	get image packages	List the packages in an image.
GET	/api/images/packages	get image packages query	List packages of the given image.

Method	URI	Name	Summary
GET	/api/packages	get packages	Search packages by id, name and/or version.
GET	/api/sources/{IDorRepoorSha}/packages	get source packages	
GET	/api/sources/packages	get source packages query	List packages of the given source.
GET	/api/vulnerabilities/{CVEID}/packages	get vulnerability packages	List packages that contain the given CVE id.

Sources

Method	URI	Name	Summary
POST	/api/sourceReport	create source report	Create a new source report. Related packages and vulnerabilities are also created.
GET	/api/packages/{IDorName}/sources	get package sources	List the sources containing the given package.
GET	/api/sources	get sources	Search for sources by ID, repository, commit sha and/or organization.
GET	/api/vulnerabilities/{CVEID}/sources	get vulnerability sources	List sources that contain the given vulnerability.

v1artifact_groups

Method	URI	Name	Summary
POST	/api/v1/artifact-groups	create artifact group	Create an artifact group with specified labels and entity
POST	/api/v1/artifact-groups/_search	search artifact groups	Query for a list of artifact group that contains image(s) with specified digests, and or source(s) with specified shas. At least one image digest or source sha must be provided. This query can be further refined by matching images and sources with a specific combination of package name and/or cve id.
POST	/api/v1/artifact-groups/vulnerabilities/_reach	search artifact groups vuln reach	Search for how many artifact groups are affected by vulnerabilities associated with the specified image(s) digests, and/or source(s) shas. At least one image digest or source sha must be provided.
POST	/api/v1/artifact-groups/vulnerabilities/_search	search artifact groups vulnerabilities	Search for all vulnerabilities associated with an artifact group that contains image(s) with specified digests, and/or source(s) with specified shas. At least one image digest or source sha must be provided.

v1images

Method	URI	Name	Summary
GET	/api/v1/images/{ID}	get image by ID	Search image by ID
GET	/api/v1/images	v1 get images	Query for images. If no parameters are given, this endpoint will return all images.

v1packages

Method	URI	Name	Summary
GET	/api/v1/packages/{ID}	get package by ID	Search package by ID
GET	/api/v1/images/packages	v1 get images packages	Query for packages with images parameters. If no parameters are given, this endpoint will return all packages related to images.
GET	/api/v1/packages	v1 get packages	Query for packages. If no parameters are given, this endpoint will return all packages.
GET	/api/v1/sources/packages	v1 get sources packages	Query for packages with source parameters. If no parameters are given, this endpoint will return all packages related to sources.

v1sources

Method	URI	Name	Summary
GET	/api/v1/sources/{ID}	get source by ID	Search source by ID
GET	/api/v1/sources	v1 get sources	Query for sources. If no parameters are given, this endpoint will return all sources.
GET	/api/v1/sources/vulnerabilities	v1 get sources vulnerabilities	Query for vulnerabilities with source parameters. If no parameters are given, this endpoint will return all vulnerabilities.

v1vulnerabilities

Method	URI	Name	Summary
GET	/api/v1/vulnerabilities/{ID}	get vulnerability by ID	Search vulnerability by ID
GET	/api/v1/images/vulnerabilities	v1 get images vulnerabilities	Query for vulnerabilities with image parameters. If no parameters are give, this endpoint will return all vulnerabilities.

vulnerabilities

Method	URI	Name	Summary
GET	/api/images/{IDorDigest}/vulnerabilities	get image vulnerabilities	List vulnerabilities from the given image.
GET	/api/packages/{IDorName}/vulnerabilities	get package vulnerabilities	List vulnerabilities from the given package.
GET	/api/sources/{IDorRepoorSha}/vulnerabilities	get source vulnerabilities	
GET	/api/sources/vulnerabilities	get source vulnerabilities query	List vulnerabilities of the given source.
GET	/api/vulnerabilities	get vulnerabilities	Search for vulnerabilities by CVE id.

Paths

Create an artifact group with specified labels and entity (*CreateArtifactGroup*)

```
POST /api/v1/artifact-groups
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
ArtifactGroupPostRequest	body	ArtifactGroupPostRequest	models.ArtifactGroupPostRequest		✓		

All responses

Code	Status	Description	Has headers	Schema
201	Created	ArtifactGroupPostResponse		schema
400	Bad Request	ErrorMessage		schema
default		ErrorMessage		schema

Responses

201 - ArtifactGroupPostResponse

Status: Created

Schema

ArtifactGroupPostResponse

400 - ErrorMessage

Status: Bad Request

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Create a new image report. Related packages and vulnerabilities are also created. (*CreateImageReport*)

```
POST /api/imageReport
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
Image	body	Image	models.Image		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Image		schema
default		ErrorMessage		schema

Responses

200 - Image

Status: OK

Schema

Image

Default Response

ErrorMessage

Schema

ErrorMessage

Create a new source report. Related packages and vulnerabilities are also created. (*CreateSourceReport*)

```
POST /api/sourceReport
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
Image	body	Source	models.Source		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Source		schema
default		ErrorMessage		schema

Responses

200 - Source

Status: OK

Schema

Source

Default Response

ErrorMessage

Schema

ErrorMessage

Search image by ID (*GetImageByID*)

```
GET /api/v1/images/{ID}
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
ID	path	uint64 (formatted integer)	uint64		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Image		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - Image

Status: OK

Schema

Image

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

List the packages in an image. (*GetImagePackages*)

```
GET /api/images/{IDorDigest}/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorDigest	path	string	string		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[]Package

Default Response

ErrorMessage

Schema

ErrorMessage

List packages of the given image. (*GetImagePackagesQuery*)

```
GET /api/images/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
digest	query	string	string				
id	query	int64 (formatted integer)	int64				
name	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[]Package

Default Response

ErrorMessage

Schema

ErrorMessage

List vulnerabilities from the given image. (*GetImageVulnerabilities*)

```
GET /api/images/{IDorDigest}/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorDigest	path	string	string		✓		
Severity	query	string	string				Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown.

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[]Vulnerability

Default Response

ErrorMessage

Schema

ErrorMessage

Search image by id, name or digest . (*GetImages*)

```
GET /api/images
```

All responses

Code	Status	Description	Has headers	Schema
200	OK	Image		schema
default		ErrorMessage		schema

Responses

200 - Image

Status: OK

Schema

Image

Default Response

ErrorMessage

Schema

ErrorMessage

Search package by ID (*GetPackageByID*)

```
GET /api/v1/packages/{ID}
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
ID	path	uint64 (formatted integer)	uint64		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

Package

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

List the images that contain the given package. (*GetPackageImages*)

```
GET /api/packages/{IDorName}/images
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorName	path	string	string		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Image		schema
default		ErrorMessage		schema

Responses

200 - Image

Status: OK

Schema

[]Image

Default Response

ErrorMessage

Schema

ErrorMessage

List the sources containing the given package. (*GetPackageSources*)

```
GET /api/packages/{IDorName}/sources
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorName	path	string	string		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Source		schema
default		ErrorMessage		schema

Responses

200 - Source

Status: OK

Schema

[]Source

Default Response

ErrorMessage

Schema

[]ErrorMessage

List vulnerabilities from the given package. (*GetPackageVulnerabilities*)

```
GET /api/packages/{IDorName}/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorName	path	string	string		✓		
Severity	query	string	string				Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown.

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[]Vulnerability

Default Response

ErrorMessage

Schema

ErrorMessage

Search packages by id, name and/or version. (*GetPackages*)

```
GET /api/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
id	query	int64 (formatted integer)	int64				Any of id or name must be provided
name	query	string	string				Any of id or name must be provided
version	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[]Package

Default Response

ErrorMessage

Schema

ErrorMessage

Search source by ID (*GetSourceByID*)

```
GET /api/v1/sources/{ID}
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
ID	path	uint64 (formatted integer)	uint64		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Source		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - Source

Status: OK

Schema

Source

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

get source packages (*GetSourcePackages*)

```
GET /api/sources/{IDorRepoorSha}/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorRepoorSha	path	string	string		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[]Package

Default Response

ErrorMessage

Schema

ErrorMessage

List packages of the given source. (*GetSourcePackagesQuery*)

```
GET /api/sources/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
id	query	uint64 (formatted integer)	uint64				
repo	query	string	string				
sha	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[]Package

Default Response

ErrorMessage

Schema

ErrorMessage

get source vulnerabilities (*GetSourceVulnerabilities*)

```
GET /api/sources/{IDorRepoorSha}/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
IDorRepoorSha	path	string	string		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[]Vulnerability

Default Response

ErrorMessage

Schema

ErrorMessage

List vulnerabilities of the given source. (*GetSourceVulnerabilitiesQuery*)

```
GET /api/sources/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
Severity	query	string	string				Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown.
id	query	uint64 (formatted integer)	uint64				
repo	query	string	string				
sha	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[]Vulnerability

Default Response

ErrorMessage

Schema

ErrorMessage

Search for sources by ID, repository, commit sha and/or organization. (*GetSources*)

```
GET /api/sources
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
id	query	int64 (formatted integer)	int64				
org	query	string	string				
repo	query	string	string				
sha	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	Source		schema
default		ErrorMessage		schema

Responses

200 - Source

Status: OK

Schema

[]Source

Default Response

ErrorMessage

Schema

ErrorMessage

Search for vulnerabilities by CVE id. (*GetVulnerabilities*)

```
GET /api/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
CVEID	query	string	string		✓		
Severity	query	string	string				Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown.

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

[]Vulnerability

Default Response

ErrorMessage

Schema

ErrorMessage

Search vulnerability by ID (*GetVulnerabilityByID*)

```
GET /api/v1/vulnerabilities/{ID}
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
ID	path	uint64 (formatted integer)	uint64		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Vulnerability		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - Vulnerability

Status: OK

Schema

Vulnerability

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

List the images that contain the given vulnerability. (*GetVulnerabilityImages*)

```
GET /api/vulnerabilities/{CVEID}/images
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
CVEID	path	string	string		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Image		schema
default		ErrorMessage		schema

Responses

200 - Image

Status: OK

Schema

[]Image

Default Response

ErrorMessage

Schema

ErrorMessage

List packages that contain the given CVE id. (*GetVulnerabilityPackages*)

```
GET /api/vulnerabilities/{CVEID}/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
CVEID	path	string	string		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Package		schema
default		ErrorMessage		schema

Responses

200 - Package

Status: OK

Schema

[]Package

Default Response

ErrorMessage

Schema

ErrorMessage

List sources that contain the given vulnerability. (*GetVulnerabilitySources*)

```
GET /api/vulnerabilities/{CVEID}/sources
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
CVEID	path	string	string		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	Source		schema
default		ErrorMessage		schema

Responses

200 - Source

Status: OK

Schema

[]Source

Default Response

ErrorMessage

Schema

ErrorMessage

health check (*HealthCheck*)

```
GET /api/health
```

All responses

Code	Status	Description	Has headers	Schema
200	OK			schema
default		ErrorMessage		schema

Responses

200

Status: OK

Schema

Default Response

ErrorMessage

Schema

ErrorMessage

Query for a list of artifact group that contains image(s) with specified digests, and or source(s) with specified shas. At least one image digest or source sha must be provided. This query can be further refined by matching images and sources with a specific combination of package name and/or cve id. (*SearchArtifactGroups*)

```
POST /api/v1/artifact-groups/_search
```

Query for a list of artifact group that contains image(s) with specified digests, and or source(s) with specified shas.

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
ArtifactGroupFilterSPostRequest	body	ArtifactGroupSearchFilters	models.ArtifactGroupSearchFilters		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedArtifactGroupResponse		schema
400	Bad Request	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - PaginatedArtifactGroupResponse

Status: OK

Schema

PaginatedArtifactGroupResponse

400 - ErrorMessage

Status: Bad Request

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Search for how many artifact groups are affected by vulnerabilities associated with the specified image(s) digests, and/or source(s) shas. At least one image digest or source sha must be provided. (*SearchArtifactGroupsVulnReach*)

```
POST /api/v1/artifact-groups/vulnerabilities/_reach
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
ArtifactGroupVulnReachFiltersPostRequest	body	ArtifactGroupVulnReachFiltersPostRequest	models.ArtifactGroupVulnReachFiltersPostRequest		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedArtifactGroupVulnReachResponse		schema
400	Bad Request	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - PaginatedArtifactGroupVulnReachResponse

Status: OK

Schema

PaginatedArtifactGroupVulnReachResponse

400 - ErrorMessage

Status: Bad Request

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Search for all vulnerabilities associated with an artifact group that contains image(s) with specified digests, and/or source(s) with

specified shas. At least one image digest or source sha must be provided. (*SearchArtifactGroupsVulnerabilities*)

```
POST /api/v1/artifact-groups/vulnerabilities/_search
```

The result can be further refined by matching the images and sources with a package name and/or an artifact group UID

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
ArtifactGroupVulnSearchFiltersPostRequest	body	ArtifactGroupVulnSearchFilters	models.ArtifactGroupVulnSearchFilters		✓		

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedArtifactGroupVulnerabilityResponse		schema
400	Bad Request	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - PaginatedArtifactGroupVulnerabilityResponse

Status: OK

Schema

PaginatedArtifactGroupVulnerabilityResponse

400 - ErrorMessage

Status: Bad Request

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Query for images. If no parameters are given, this endpoint will return all images. (*V1GetImages*)

```
GET /api/v1/images
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
all	query	boolean	bool				If no pagination parameters are provided, defaults to true and returns all available results.
digest	query	string	string				
name	query	string	string				
page	query	int64 (formatted integer)	int64			1	
page_size	query	int64 (formatted integer)	int64			20	
registry	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedImageResponse		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - PaginatedImageResponse

Status: OK

Schema

PaginatedImageResponse

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Query for packages with images parameters. If no parameters are given, this endpoint will return all packages related to images. (*V1GetImagesPackages*)

```
GET /api/v1/images/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
all	query	boolean	bool				If no pagination parameters are provided, defaults to true and returns all available results.
digest	query	string	string				
name	query	string	string				
package_name	query	string	string				Substring package name filter. For example, setting `name=cur` would match `curl` and `libcurl`.
page	query	int64 (formatted integer)	int64			1	
page_size	query	int64 (formatted integer)	int64			20	
registry	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedPackageResponse		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - PaginatedPackageResponse

Status: OK

Schema

PaginatedPackageResponse

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Query for vulnerabilities with image parameters. If no parameters are give, this endpoint will return all vulnerabilities. (*V1GetImagesVulnerabilities*)

```
GET /api/v1/images/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
Severity	query	string	string				Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown.
all	query	boolean	bool				If no pagination parameters are provided, defaults to true and returns all available results.
digest	query	string	string				
name	query	string	string				
page	query	int64 (formatted integer)	int64			1	
page_size	query	int64 (formatted integer)	int64			20	
registry	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedVulnerabilityResponse		schema
404	Not Found	ErrorMessage		schema

Code	Status	Description	Has headers	Schema
default		ErrorMessage		schema

Responses

200 - PaginatedVulnerabilityResponse

Status: OK

Schema

PaginatedVulnerabilityResponse

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Query for packages. If no parameters are given, this endpoint will return all packages. (*V1GetPackages*)

```
GET /api/v1/packages
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
all	query	boolean	bool				If no pagination parameters are provided, defaults to true and returns all available results.
name	query	string	string				Name filter works as a substring match on the package name. For example, setting `name=cur` would match `curl` and `libcurl`.
package_manager	query	string	string				
page	query	int64 (formatted integer)	int64			1	

Name	Source	Type	Go type	Separator	Required	Default	Description
page_size	query	int64 (formatted integer)	int64			20	
version	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedPackageResponse		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - PaginatedPackageResponse

Status: OK

Schema

PaginatedPackageResponse

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Query for sources. If no parameters are given, this endpoint will return all sources. (*V1GetSources*)

```
GET /api/v1/sources
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
all	query	boolean	bool				If no pagination parameters are provided, defaults to true and returns all available results.
org	query	string	string				
page	query	int64 (formatted integer)	int64			1	
page_size	query	int64 (formatted integer)	int64			20	
repo	query	string	string				
sha	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedSourceResponse		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - PaginatedSourceResponse

Status: OK

Schema

PaginatedSourceResponse

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Query for packages with source parameters. If no parameters are given, this endpoint will return all packages related to sources. (*V1GetSourcesPackages*)

```
GET /api/v1/sources/packages
```

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedPackageResponse		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - PaginatedPackageResponse

Status: OK

Schema

PaginatedPackageResponse

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Query for vulnerabilities with source parameters. If no parameters are given, this endpoint will return all vulnerabilities. (*V1GetSourcesVulnerabilities*)

```
GET /api/v1/sources/vulnerabilities
```

Parameters

Name	Source	Type	Go type	Separator	Required	Default	Description
Severity	query	string	string				Case insensitive vulnerabilities severity filter. Possible values are: low, medium, high, critical, unknown.
all	query	boolean	bool				If no pagination parameters are provided, defaults to true and returns all available results.
org	query	string	string				
page	query	int64 (formatted integer)	int64			1	
page_size	query	int64 (formatted integer)	int64			20	
repo	query	string	string				
sha	query	string	string				

All responses

Code	Status	Description	Has headers	Schema
200	OK	PaginatedVulnerabilityResponse		schema
404	Not Found	ErrorMessage		schema
default		ErrorMessage		schema

Responses

200 - PaginatedVulnerabilityResponse

Status: OK

Schema

PaginatedVulnerabilityResponse

404 - ErrorMessage

Status: Not Found

Schema

ErrorMessage

Default Response

ErrorMessage

Schema

ErrorMessage

Models

ArtifactGroupPostRequest

Properties

Name	Type	Go type	Required	Default	Description	Example
EntityID	uint64 (formatted integer)	`uint64`			The database ID of the source or image being associated with this artifact group	`24`
Labels	map of string	`map[string]string`			Key-Value pair of labels associated with the Artifact Group	`{"env":"production","namespace":"default"}`
Type	string	`string`			The entity type being associated with this artifact group. Allowable values: image, source	`image`
UID	string	`string`	✓		Unique identifier for the Artifact Group such as workload UID	`8b1cc5da-fabe-45a6-ab8c-49260bbeef99`

ArtifactGroupResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Entities	[]Entity	`[]*Entity`			Entities associated with the Artifact Group	
Labels	map of string	`map[string]string`			Key-Value pair of labels associated with the Artifact Group	`{"env":"production","namespace":"default"}`
UID	string	`string`			Unique identifier for the Artifact Group such as workload UID	`8b1cc5da-fabe-45a6-ab8c-49260bbeef99`

ArtifactGroupSearchFilters

Properties

Name	Type	Go type	Required	Default	Description	Example
All	boolean	`bool`			If no pagination parameters are provided, defaults to true and returns all available results.	
CVEID	string	`string`			An optional CVE ID that the image and source must contain. Only packages, and their images and sources, with this CVE ID will be returned. If both package name and CVE ID are provided, then only the images and sources with the specified package name and CVE ID will be returned.	`CVE-7467-2020`

Name	Type	Go type	Required	Default	Description	Example
Digests	[]string	`[]string`			A list of image digests. At least one image digest or source sha must be provided.	`["9n38274ods897fmay487gsdyfga678wr82","7n38274ods897fmay487gsdyfga678wr82"]`
PackageName	string	`string`			An optional package name that the image and source must contain. Only packages, and their images and sources, with this name will be returned. If both package name and CVE ID are provided, then only the images and sources with the specified package name and CVE ID will be returned.	`package1`
Page	int64 (formatted integer)	`int64`		`1`		
PageSize	int64 (formatted integer)	`int64`		`20`		
Shas	[]string	`[]string`			A list of source shas. At least one image digest or source sha must be provided.	`["sha256:2c11624a8d9c9071996a886a4acaf09939ef3386e4c07735c6a2532f02eed4ea","sha256:04bafe0d8df23ec342edb72acc3fb02f61c418bc6e8d7093149956a9aad2d12a"]`

ArtifactGroupVulnReachFiltersPostRequest

Properties

Name	Type	Go type	Required	Default	Description	Example
All	boolean	`bool`			If no pagination parameters are provided, defaults to true and returns all available results.	
Digests	[]string	`[]string`			A list of image digests.	`["sha256:2c11624a8d9c9071996a886a4acaf09939ef3386e4c07735c6a2532f02eed4ea","sha256:04bafe0d8df23ec342edb72acc3fb02f61c418bc6e8d7093149956a9aad2d12a"]`
Page	int64 (formatted integer)	`int64`		`1`		
PageSize	int64 (formatted integer)	`int64`		`20`		

Name	Type	Go type	Required	Default	Description	Example
Shas	[]string	`[]string`			A list of source shas.	`["9n38274ods897fmay487gsdyfga678wr82","7n38274ods897fmay487gsdyfga678wr82"]`

ArtifactGroupVulnReachPostResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
AgCount	uint64 (formatted integer)	`uint64`			Number of artifact groups affected by the vulnerability	`5`
Vulnerability	VulnResponse	`VulnResponse`				

ArtifactGroupVulnSearchFilters

Properties

Name	Type	Go type	Required	Default	Description	Example
All	boolean	`bool`			If no pagination parameters are provided, defaults to true and returns all available results.	
ArtifactGroupUID	string	`string`			An optional artifact group UID that the image and source must contain. Only artifact groups, and their images and sources, with this artifact group UID will be returned. If both package name and artifact group UID are provided, then only the images and sources with the specified package name and artifact group UID will be returned.	`9aa3548e-5fae-11ed-9b6a-0242ac120002`
Digests	[]string	`[]string`			A list of image digests. At least one image digest or source sha must be provided.	`["9n38274ods897fmay487gsdyfga678wr82","7n38274ods897fmay487gsdyfga678wr82"]`
PackageName	string	`string`			An optional package name that the image and source must contain. Only packages, and their images and sources, with this name will be returned. If both package name and artifact group UID are provided, then only the images and sources with the specified package name and artifact group UID will be returned.	`package1`
Page	int64 (formatted integer)	`int64`		`1`		
PageSize	int64 (formatted integer)	`int64`		`20`		

Name	Type	Go type	Required	Default	Description	Example
Shas	[]string	`[]string`			A list of source shas. At least one image digest or source sha must be provided.	`["sha256:2c11624a8d9c9071996a886a4acaf09939ef3386e4c07735c6a2532f02eed4ea","sha256:04bafe0d8df23ec342edb72acc3fb02f61c418bc6e8d7093149956a9aad2d12a"]`

DeletedAt

- composed type NullTime

Entity

Properties

Name	Type	Go type	Required	Default	Description	Example
Digest	string	`string`			The digest of the image entity. Only visible if the entity is of image type	`sha256:f7de1564f13da1ef7e5720ebce14006793242c0d8d7d60c343632bcf3bc5306d`
Host	string	`string`			The dns name where the source entity is hosted on. Only visible if the entity type is of source type	`gitlab.com`
ID	uint64 (formatted integer)	`uint64`	✓		The database ID of the source or image	`24`
Name	string	`string`			The name of the image entity. Only visible if the entity is of image type.	`checkr/flagr`
Org	string	`string`			The organization name of the source entity. Only visible if the entity type is of source type	`my-organization`
Packages	[]Package	`[]*Package`				
Registry	string	`string`			The DNS name of the registry that stores the image entity. Only visible if the entity is of image type	`docker.io`
Repo	string	`string`			The repository name of the source entity. Only visible if the entity type is of source type	`my-sample-repo`
Sha	string	`string`			The commit sha of the source entity. Only visible if the entity type is of source type	`d6cd1e2bd19e03a81132a23b2025920577f84e37`
Type	string	`string`	✓		The entity Type of scan that is stored. This is set to either “image” or “source”.	`image`

ErrorMessage

ErrorMessage wraps an error message in a struct so responses are properly marshalled as a JSON object.

Properties

Name	Type	Go type	Required	Default	Description	Example
Message	string	`string`			in: body	`something went wrong`

Image

Properties

Name	Type	Go type	Required	Default	Description	Example
Digest	string	`string`	✓			`9n38274ods897fmay487gsdyfga678wr82`
ID	uint64 (formatted integer)	`uint64`				
Name	string	`string`	✓			`myorg/application`
Packages	[]Package	`[]*Package`				
Registry	string	`string`	✓			`docker.io`
Sources	[]Source	`[]*Source`				

MethodType

Properties

Name	Type	Go type	Required	Default	Description	Example
CreatedAt	date-time (formatted string)	`strfmt.DateTime`				
DeletedAt	DeletedAt	`DeletedAt`				
ID	uint64 (formatted integer)	`uint64`				
Name	string	`string`				
Rating	[]Rating	`[]*Rating`				
UpdatedAt	date-time (formatted string)	`strfmt.DateTime`				

Model

Model a basic GoLang struct which includes the following fields: ID, CreatedAt, UpdatedAt, DeletedAt It may be embedded into your model or you may build your own model without it type User struct { gorm.Model }

Properties

Name	Type	Go type	Required	Default	Description	Example
CreatedAt	date-time (formatted string)	`strfmt.DateTime`				
DeletedAt	DeletedAt	`DeletedAt`				
ID	uint64 (formatted integer)	`uint64`				

Name	Type	Go type	Required	Default	Description	Example
UpdatedAt	date-time (formatted string)	`strfmt.DateTime`				

NullTime

NullTime implements the Scanner interface so it can be used as a scan destination, similar to NullString.

Properties

Name	Type	Go type	Required	Default	Description	Example
Time	date-time (formatted string)	`strfmt.DateTime`				
Valid	boolean	`bool`				

Package

Properties

Name	Type	Go type	Required	Default	Description	Example
Homepage	string	`string`				
ID	uint64 (formatted integer)	`uint64`				
Images	[]Image	`[]*Image`				
Name	string	`string`				
PackageManager	string	`string`				
Sources	[]Source	`[]*Source`				
Version	string	`string`				
Vulnerabilities	[]Vulnerability	`[]*Vulnerability`				

PaginatedArtifactGroupVulnReachResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Count	int64 (formatted integer)	`int64`				`10`
CurrentPage	int64 (formatted integer)	`int64`				`1`
LastPage	int64 (formatted integer)	`int64`				`2`
PageSize	int64 (formatted integer)	`int64`				`20`
Results	[]ArtifactGroupVulnReachPostResponse	`[]*ArtifactGroupVulnReachPostResponse`				

PaginatedResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Count	int64 (formatted integer)	`int64`				`10`

Name	Type	Go type	Required	Default	Description	Example
CurrentPage	int64 (formatted integer)	`int64`				`1`
LastPage	int64 (formatted integer)	`int64`				`2`
PageSize	int64 (formatted integer)	`int64`				`20`
Results	[]interface{}	`[]interface{}`				

Rating

Properties

Name	Type	Go type	Required	Default	Description	Example
ID	uint64 (formatted integer)	`uint64`				
MethodType	MethodType	`MethodType`				
MethodTypeID	uint64 (formatted integer)	`uint64`				
Score	double (formatted number)	`float64`				
Severity	string	`string`				
Vector	string	`string`				

RatingResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
ID	uint64 (formatted integer)	`uint64`			Rating ID	`3`
Score	double (formatted number)	`float64`			CVSS score	`9.7`
Severity	string	`string`			Threat level of vulnerability	`High`
Vector	string	`string`			CVSS score in vector format	`AV:L/AC:L/Au:N/C:C/I:C/A:C`

Source

Properties

Name	Type	Go type	Required	Default	Description	Example
DeletedAt	DeletedAt	`DeletedAt`				
Host	string	`string`				`gitlab.com`
ID	uint64 (formatted integer)	`uint64`				
Images	[]Image	`[]*Image`				
Organization	string	`string`				`vmware`
Packages	[]Package	`[]*Package`				
Repository	string	`string`	✓			`myproject`

Name	Type	Go type	Required	Default	Description	Example
Sha	string	`string`	✓			`0eb5fcd1`

StringArray

[]string

VulnResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
CNA	string	`string`			CVE Numbering Authority	`GitHub, Inc.`
CVEID	string	`string`			CVE ID of the vulnerability	`CVE-7467-2020`
Description	string	`string`			Description of the vulnerability	`IBM Datapower Gateway 10.0.2.0 through 10.0.4.0, 10.0.1.0 through 10.0.1.5, and 2018.4.1.0 through 2018.4.1.18 could allow unauthorized viewing of logs and files due to insufficient authorization checks. IBM X-Force ID: 218856.`
ID	uint64 (formatted integer)	`uint64`			Vulnerability ID	`12`
Ratings	[]Rating Response	`[]*RatingResponse`			Rating information	
References	[]string	`[]string`			Additional external links	`https://github.com/example/repo/issues/11`
URL	string	`string`			Related url to the vulnerability	`https://nvd.nist.gov/vuln/detail/CVE-7467-2020`

Vulnerability

Properties

Name	Type	Go type	Required	Default	Description	Example
CNA	string	`string`				`GitHub, Inc.`
CVEID	string	`string`	✓			`CVE-7467-2020`
Description	string	`string`				`A description of CVE-7467-2020`
ID	uint64 (formatted integer)	`uint64`				
Packages	[]Package	`[]*Package`				

Name	Type	Go type	Required	Default	Description	Example
Ratings	[]Rating	`[]*Rating`				
References	StringArray	`StringArray`				
URL	string	`string`				`https://nvd.nist.gov/vuln/detail/CVE-7467-2020`

artifactGroupPostEntity

Properties

Name	Type	Go type	Required	Default	Description	Example
ID	uint64 (formatted integer)	`uint64`	✓		The database ID of the source or image	`24`
Type	string	`string`	✓		The entity Type of scan that is stored. This is set to either “image” or “source”.	`image`

artifactGroupPostResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Entities	[]ArtifactGroupPostEntity	`[]*ArtifactGroupPostEntity`			Entities associated with the Artifact Group	
Labels	map of string	`map[string]string`			Key-Value pair of labels associated with the Artifact Group	`{"env":"production", "namespace":"default"}`
UID	string	`string`			Unique identifier for the Artifact Group such as workload UID	`8b1cc5da-fabe-45a6-ab8c-49260bbeef99`

artifactGroupVulnArtifactGroup

Properties

Name	Type	Go type	Required	Default	Description	Example
Entities	[]ArtifactGroupVulnEntity	`[]*ArtifactGroupVulnEntity`			Entities associated with the Artifact Group	
Labels	map of string	`map[string]string`			Key-Value pair of labels associated with the Artifact Group	`{"env":"production", "namespace":"default"}`
UID	string	`string`			Unique identifier for the Artifact Group such as workload UID	`8b1cc5da-fabe-45a6-ab8c-49260bbeef99`

artifactGroupVulnEntity

Properties

Name	Type	Go type	Required	Default	Description	Example
Digest	string	`string`			The digest of the image entity. Only visible if the entity is of image type	`sha256:f7de1564f13da1ef7e5720ebce14006793242c0d8d7d60c343632bcf3bc5306d`
Host	string	`string`			The dns name where the source entity is hosted on. Only visible if the entity type is of source type	`gitlab.com`
ID	uint64 (formatted integer)	`uint64`	✓		The database ID of the source or image	`24`
Name	string	`string`			The name of the image entity. Only visible if the entity is of image type.	`checkr/flagr`
Org	string	`string`			The organization name of the source entity. Only visible if the entity type is of source type	`my-organization`
Registry	string	`string`			The DNS name of the registry that stores the image entity. Only visible if the entity is of image type	`docker.io`
Repo	string	`string`			The repository name of the source entity. Only visible if the entity type is of source type	`my-sample-repo`
Sha	string	`string`			The commit sha of the source entity. Only visible if the entity type is of source type	`d6cd1e2bd19e03a81132a23b2025920577f84e37`
Type	string	`string`	✓		The entity Type of scan that is stored. This is set to either “image” or “source”.	`image`

artifactGroupVulnPackage

Properties

Name	Type	Go type	Required	Default	Description	Example
Homepage	string	`string`				
ID	uint64 (formatted integer)	`uint64`				
Images	[]Image	`[]*Image`			This field will always be empty	`[]`
Name	string	`string`				
PackageManager	string	`string`				
Sources	[]Source	`[]*Source`			This field will always be empty	`[]`
Version	string	`string`				
Vulnerabilities	[]Vulnerability	`[]*Vulnerability`			This field will always be empty	`[]`

artifactGroupVulnResult

Properties

Name	Type	Go type	Required	Default	Description	Example
ArtifactGroups	[]ArtifactGroupVulnArtifactGroup	`[]*ArtifactGroupVulnArtifactGroup`				
CNA	string	`string`				`GitHub, Inc.`
CVEID	string	`string`	✓			`CVE-7467-2020`
Description	string	`string`				`A description of CVE-7467-2020`
ID	uint64 (formatted integer)	`uint64`				
Packages	[]ArtifactGroupVulnPackage	`[]*ArtifactGroupVulnPackage`				
Ratings	[]Rating	`[]*Rating`				
References	StringArray	`StringArray`				
URL	string	`string`				`https://nvd.nist.gov/vuln/detail/CVE-7467-2020`

paginatedArtifactGroupResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Count	int64 (formatted integer)	`int64`				`10`
CurrentPage	int64 (formatted integer)	`int64`				`1`
LastPage	int64 (formatted integer)	`int64`				`2`
PageSize	int64 (formatted integer)	`int64`				`20`
Results	[]ArtifactGroupResponse	`[]*ArtifactGroupResponse`				

paginatedArtifactGroupVulnerabilityResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Count	int64 (formatted integer)	`int64`				`10`
CurrentPage	int64 (formatted integer)	`int64`				`1`
LastPage	int64 (formatted integer)	`int64`				`2`
PageSize	int64 (formatted integer)	`int64`				`20`
Results	[]ArtifactGroupVulnResult	`[]*ArtifactGroupVulnResult`				

paginatedImageResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Count	int64 (formatted integer)	`int64`				`10`
CurrentPage	int64 (formatted integer)	`int64`				`1`
LastPage	int64 (formatted integer)	`int64`				`2`
PageSize	int64 (formatted integer)	`int64`				`20`
Results	[]ResponseImage	`[]*ResponseImage`				

paginatedPackageResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Count	int64 (formatted integer)	`int64`				`10`
CurrentPage	int64 (formatted integer)	`int64`				`1`
LastPage	int64 (formatted integer)	`int64`				`2`
PageSize	int64 (formatted integer)	`int64`				`20`
Results	[]ResponsePackage	`[]*ResponsePackage`				

paginatedSourceResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Count	int64 (formatted integer)	`int64`				`10`
CurrentPage	int64 (formatted integer)	`int64`				`1`
LastPage	int64 (formatted integer)	`int64`				`2`
PageSize	int64 (formatted integer)	`int64`				`20`
Results	[]ResponseSource	`[]*ResponseSource`				

paginatedVulnerabilityResponse

Properties

Name	Type	Go type	Required	Default	Description	Example
Count	int64 (formatted integer)	`int64`				`10`
CurrentPage	int64 (formatted integer)	`int64`				`1`
LastPage	int64 (formatted integer)	`int64`				`2`
PageSize	int64 (formatted integer)	`int64`				`20`
Results	[]ResponseVulnerability	`[]*ResponseVulnerability`				

responseImage

Properties

Name	Type	Go type	Required	Default	Description	Example
CreatedAt	date-time (formatted string)	`strfmt.DateTime`				
Digest	string	`string`	✓			`9n38274ods897fmay487gsdyfga678wr82`
ID	uint64 (formatted integer)	`uint64`				
Name	string	`string`	✓			`myorg/application`
Packages	[]Package	`[]*Package`				
Registry	string	`string`	✓			`docker.io`
Sources	[]Source	`[]*Source`				
Updated At	date-time (formatted string)	`strfmt.DateTime`				

responsePackage

Properties

Name	Type	Go type	Required	Default	Description	Example
CreatedAt	date-time (formatted string)	`strfmt.DateTime`				
Homepage	string	`string`				
ID	uint64 (formatted integer)	`uint64`				
Images	[]Image	`[]*Image`				
Name	string	`string`				
PackageManager	string	`string`				
Sources	[]Source	`[]*Source`				
UpdatedAt	date-time (formatted string)	`strfmt.DateTime`				
Version	string	`string`				
Vulnerabilities	[]Vulnerability	`[]*Vulnerability`				

responseSource

Properties

Name	Type	Go type	Required	Default	Description	Example
CreatedAt	date-time (formatted string)	`strfmt.DateTime`				
DeletedAt	DeletedAt	`DeletedAt`				
Host	string	`string`				`gitlab.com`
ID	uint64 (formatted integer)	`uint64`				
Images	[]Image	`[]*Image`				
Organization	string	`string`				`vmware`
Packages	[]Package	`[]*Package`				
Repository	string	`string`	✓			`myproject`

Name	Type	Go type	Required	Default	Description	Example
Sha	string	`string`	✓			`0eb5fcd1`
UpdatedAt	date-time (formatted string)	`strfmt.DateTime`				

responseVulnerability

Properties

Name	Type	Go type	Required	Default	Description	Example
CNA	string	`string`				`GitHub, Inc.`
CVEID	string	`string`	✓			`CVE-7467-2020`
CreatedAt	date-time (formatted string)	`strfmt.DateTime`				
Description	string	`string`				`A description of CVE-7467-2020`
ID	uint64 (formatted integer)	`uint64`				
Packages	[]Package	`[]*Package`				
Ratings	[]Rating	`[]*Rating`				
References	StringArray	`StringArray`				
URL	string	`string`				`https://nvd.nist.gov/vuln/detail/CVE-7467-2020`
UpdatedAt	date-time (formatted string)	`strfmt.DateTime`				

API walkthrough for Supply Chain Security Tools - Store

This topic tells you how to make an API call that you can use with Supply Chain Security Tools (SCST) - Store. For information about using the SCST - Store API, see API reference for Supply Chain Security Tools - Store.

Use curl to post an image report

This procedure uses Ingress, if Tanzu Application Platform is deployed without Ingress, see Use your NodePort with Supply Chain Security Tools - Store and Use your LoadBalancer with Supply Chain Security Tools - Store. Complete the following steps:

1. Switch to the kubectl context or kubeconfig to target the View cluster.
2. Retrieve the CA certificate and store it locally. Run:

```
kubectl get secret ingress-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | base64 -d > /tmp/ca.crt
```

3. Using the `health` endpoint as an example, run:

```
curl -i https://metadata-store.INGRESS-DOMAIN/api/HEALTH \
   --cacert /tmp/ca.crt
```

For example:

```
$ curl -i https://metadata-store.example.com/api/health \
  --cacert /tmp/ca.crt

HTTP/2 200
content-length: 0
date: Tue, 23 Jan 2024 22:50:57 GMT
x-envoy-upstream-service-time: 0
server: envoy
```

4. To make a request to an authenticated endpoint an access token is required. To retrieve the `metadata-store-read-write-client` access token, run:

   ```
   export METADATA_STORE_ACCESS_TOKEN=$(kubectl get secrets metadata-store-read-wr
   ite-client -n metadata-store -o jsonpath="{.data.token}" | base64 -d)
   ```

   For more information, see Retrieve access tokens for Supply Chain Security Tools - Store.

5. Using the `api/imageReport` endpoint as an example, create a post request:

   ```
   curl https://metadata-store.INGRESS-DOMAIN/API/IMAGE-REPORT \
       --cacert /tmp/ca.crt \
       -H "Authorization: Bearer ${METADATA_STORE_ACCESS_TOKEN}" \
       -H "Content-Type: application/json" \
       -X POST \
       --data "@ABSOLUTE-PATH-TO-THE-POST-BODY"
   ```

   Where `ABSOLUTE-PATH-TO-THE-POST-BODY` is the absolute filepath of the API JSON for an image report.

   For example, the following is a sample post body of an image report API JSON:

   ```
   {
     "Name" : "burger-image-2",
     "Registry" : "test-registry",
     "Digest" : "test-digest@45asd61asasssdfsdfddssghjkdfsdfasdfasdsdasdassdfghjdd
   asfddfsadfadfgfshdasdfsdfsdfsdasdsdfsdfadsdassdfdasdfaasdsdfsddfsdasgsasddffdgf
   dasddfgdfssdfakasdasdasdsdasddasdsd23",
     "Sources" : [
       {
         "Repository" : "aaaaoslfdfggo",
         "Organization" : "pivotal",
         "Sha" : "1235assdfssadfacfddxdf41",
         "Host" : "http://oslo.io",
         "Packages" : [
           {
             "Name" : "Source package5",
             "Version" : "v2sfsfdd34",
             "PackageManager" : "test-manager",
             "Vulnerabilities" : [
               {
                 "CVEID" : "0011",
                 "PrimaryURL" : "http://www.mynamejeff.comm",
                 "Description" : "Bye",
                 "CNA" : "NVD",
                 "Ratings": [{
                   "Vector" : "AV:L/AC:L/Au:N/C:P/I:P/A:P",
                   "Score" : 0,
                   "MethodTypeID" : 1,
                   "Severity":   "High"
                 }],
                 "References" : [""]
               }
   ```

```
          ]
        }
      ]
    }
  ],
  "Packages" : [
    {
      "Name" : "bob-dependency-35daasds56j",
      "Version" : "v2",
      "PackageManager" : "test-manager",
      "Vulnerabilities" : [
        {
          "CVEID" : "002",
          "PrimaryURL" : "http://www.mynamejeff.comm",
          "Description" : "Bye",
          "CNA" : "NVD",
          "Ratings": [{
            "Vector" : "AV:L/AC:L/Au:N/C:P/I:P/A:P",
            "Score" : 0,
            "MethodTypeID" : 1,
            "Severity":   "High"
          }],
          "References" : [""]
        }
      ]
    }
  ]
}
```

Connect to the PostgreSQL database

You can use a PostgreSQL database with Supply Chain Security Tools - Store. To connect to the PostgreSQL database, you need the following values:

- database name
- user name
- password
- database host
- database port
- database CA certificate

Connect to the PostgreSQL database:

1. Obtain the database name, user name, password, and CA certificate. Run:

   ```
   db_name=$(kubectl get secret postgres-db-secret -n metadata-store -o json | jq
   -r '.data.POSTGRES_DB' | base64 -d)
   db_username=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
   jq -r '.data.POSTGRES_USER' | base64 -d)
   db_password=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
   jq -r '.data.POSTGRES_PASSWORD' | base64 -d)

   db_ca_dir=$(mktemp -d -t ca-cert-XXXX)
   db_ca_path="$db_ca_dir/ca.crt"
   kubectl get secrets postgres-db-tls-cert -n metadata-store -o json | jq -r '.da
   ta."ca.crt"' | base64 -d > $db_ca_path
   ```

   If the password was auto-generated, the `password` command returns an empty string. Run:

   ```
   db_password=$(kubectl get secret postgres-db-password -n metadata-store -o json
   ```

```
| jq -r '.data.DB_PASSWORD' | base64 -d)
```

2. In a separate terminal, run:

```
kubectl port-forward service/metadata-store-db 5432:5432 -n metadata-store
```

3. Set the database host and port values on the first terminal:

```
db_host="localhost"
db_port=5432
```

4. To port forward to a different local port number, use the following command template:

```
kubectl port-forward service/metadata-store-db <LOCAL_PORT>:5432 -n metadata-st
ore
```

   Where `LOCAL-PORT` is the port number for the database you want to use.

You can now connect to the database and make queries. For example:

```
psql "host=$db_host port=$db_port user=$db_username dbname=$db_name sslmode=verify-ca
sslrootcert=$db_ca_path" -c "SELECT * FROM images"
```

You can use GUI clients such as Postico or DBeaver to interact with the database.

Deployment details and configuration for Supply Chain Security Tools - Store

This topic describes how you can deploy and configure your Kubernetes cluster for Supply Chain Security Tools (SCST) - Store.

What is deployed

The installation creates the following in your Kubernetes cluster:

- Two components — an API back end and a database. Each component includes:
  - service
  - deployment
  - replicaset
  - Pod
- Persistent volume claim
- External IP address (based on a deployment configuration set to use `LoadBalancer`).
- A Kubernetes secret to allow pulling SCST - Store images from a registry.
- A namespace called `metadata-store`.
- A service account with read-write privileges named `metadata-store-read-write-client`, and a corresponding secret for the service account. It's bound to a ClusterRole named `metadata-store-read-write`.
- A read-only ClusterRole named `metadata-store-read-only` that isn't bound to a service account. See Service Accounts.
- (Optional) An HTTPProxy object for ingress support.

Deployment configuration

All configurations are nested inside of `metadata_store` in your tap values deployment YAML.

Supported Network Configurations

The following connection methods are recommended based on Tanzu Application Platform setup:

- Single or multicluster with Contour = `Ingress`
- Single cluster without Contour and with `LoadBalancer` support = `LoadBalancer`
- Single cluster without Contour and without `LoadBalancer` = `NodePort`
- Multicluster without Contour = Not supported

For a production environment, VMware recommends that you install SCST - Store with ingress enabled.

App service type

Supported values include `LoadBalancer`, `ClusterIP`, `NodePort`. The `app_service_type` is set to `LoadBalancer` by default. If your environment does not support `LoadBalancer`, and you want to use `ClusterIP`, configure the `app_service_type` property in your deployment YAML:

```
app_service_type: "ClusterIP"
```

If you set the `ingress_enabled` to `"true"`, VMware recommends setting the `app_service_type` property to `"ClusterIP"`.

Ingress support

SCST - Store's values file allows you to enable ingress support and to configure a custom domain name to use Contour to provide external access to SCST - Store's API. For example:

```
ingress_enabled: "true"
ingress_domain: "example.com"
app_service_type: "ClusterIP" # recommended setting
```

An HTTPProxy object is installed with `metadata-store.example.com` as the fully qualified domain name. See Ingress.

Note

The `ingress_enabled` property expects a string value of `"true"` or `"false"`, not a Boolean value.

Database configuration

The default database included with the deployment is meant to get users started using the metadata store. The default database deployment does not support many enterprise production requirements, including scaling, redundancy, or failover. However, it is a secure deployment.

Using AWS RDS PostgreSQL database

Users can also configure the deployment to use their own RDS database instead of the default. See AWS RDS Postgres Configuration.

Using external PostgreSQL database

Users can configure the deployment to use any other PostgreSQL database. See Use external postgres database.

Custom database password

By default, a database password is generated upon deployment. To configure a custom password, use the `db_password` property in the deployment YAML.

```
db_password: "PASSWORD-0123"
```

Where `PASSWORD-0123` is the same password used between deployments.

Important

There is a known issue related to changing database passwords Persistent Volume Retains Data.

Service accounts

By default, a service account with read-write privileges to the metadata store app is installed. This service account is a cluster-wide account that uses ClusterRole. If you don't want the service account and role, set the `add_default_rw_service_account` property to `"false"`. To create a custom service account, see Create Service Account.

The store creates a read-only cluster role, which is bound to a service account by using `ClusterRoleBinding`. To create service accounts to bind to this cluster role, see Create Service Account.

Exporting certificates

SCST - Store creates a Secret Export for exporting certificates to `Supply Chain Security Tools - Scan` to securely post scan results. These certificates are exported to the namespace where `Supply Chain Security Tools - Scan` is installed.

Configure your AWS RDS PostgreSQL configuration

This topic describes how you can configure your AWS RDS PostgreSQL configuration for Supply Chain Security Tools (SCST) - Store.

Prerequisites

- AWS Account

Setup certificate and configuration

1. Create an Amazon RDS Postgres using the Amazon RDS Getting Started Guide
2. Once the database instance starts, retrieve the following information:
    1. DB Instance Endpoint
    2. Master Username
    3. Master Password
    4. Database Name

> **Note**
>
> If the database name is `-` in the AWS RDS UI, the value is likely `postgres`.

3. Create a security group to allow inbound connections from the cluster to the Postgres DB
4. Retrieve the corresponding CA Certificate that signed the Postgres TLS Certificate using the following link
5. In the `metadata-store-values.yaml` fill the following settings:

```
db_host: "<DB Instance Endpoint>"
db_user: "<Master Username>"
db_password: "<Master Password>"
db_name: "<Database Name>"
db_port: "5432"
db_sslmode: "verify-full"
db_max_open_conns: 10
db_max_idle_conns: 100
db_conn_max_lifetime: 60
db_ca_certificate: |
  <Corresponding CA Certification>
  ...
  ...
  ...
deploy_internal_db: "false"
```

> **Note**
>
> If `deploy_internal_db` is set to `false`, an instance of Postgres will not be deployed in the cluster.

Use external PostgreSQL database for Supply Chain Security Tools - Store

This topic describes how you can configure and use your external PostgreSQL database for Supply Chain Security Tools (SCST) - Store.

Prerequisites

- Set up your external PostgreSQL database. After the database instance starts, retrieve the following information:
    1. Database Instance Endpoint
    2. Main User name
    3. Main Password
    4. Database Name

Set up certificate and configuration

1. Create a security group to allow inbound connections from the cluster to the PostgreSQL database.
2. Retrieve the corresponding CA Certificate that signed the PostgreSQL TLS Certificate.

3. In the `metadata-store-values.yaml` fill the following settings:

```
db_host: "<DB Instance Endpoint>"
db_user: "<Master Username>"
db_password: "<Master Password>"
db_name: "<Database Name>"
db_port: "5432"
db_sslmode: "verify-full"
db_max_open_conns: 10
db_max_idle_conns: 100
db_conn_max_lifetime: 60
db_ca_certificate: |
  <Corresponding CA Certification>
  ...
  ...
  ...
deploy_internal_db: "false"
```

Note

If `deploy_internal_db` is set to `false`, an instance of PostgreSQL is not deployed in the cluster.

Validation

Verification was done using bitnami PostgreSQL. You can get more information from the bitnami documentation.

Database backup recommendations for Supply Chain Security Tools - Store

This topic describes database backup recommendations for Supply Chain Security Tools - Store.

By default, the metadata store uses a `PersistentVolume` mounted on a Postgres instance, making it a stateful component of Tanzu Application Platform. VMware recommends implementing a regular backup strategy as part of your disaster recovery plan when using the provided Postgres instance.

Backup

You can use Velero to create regular backups.

Note

Backup support for `PersistentVolume` depends on the used `StorageClass` and existing provider plug-ins. See the officially supported plug-ins here.

```
velero install --provider PROVIDER --bucket BUCKET-NAME --plugins PLUGIN-IMAGE-LOCATIO
N --secret-file SECRET-FILE
```

Where:

- `PROVIDER` is the name of the provider you want to use.
- `BUCKET-NAME` is the name of the bucket you want to use.
- `PLUGIN-IMAGE-LOCATION` is the location of the plug ins you want to use.

- `SECRET-FILE` is the file where the secret is located.

Velero CLI can then be used to create a backup of all the resources in the `metadata-store` namespace, including `PersistentVolumeClaim` and `PersistentVolume`.

```
velero backup create metadata-store-$(date '+%s') --include-namespaces=metadata-store
```

Restore

Velero CLI can restore the Store in the same or a different cluster. The same namespace can be used to restore, but may collide with other Supply Chain Security Tools – Store installations. Furthermore, restoring into the same namespace restores a fully functional instance of Supply Chain Security Tools – Store; however, this instance is not managed by Tanzu Application Platform and can cause conflicts with future installations.

```
velero restore create restore-metadata-store-$timestamp --from-backup metadata-store
-$timestamp --namespace-mappings metadata-store:metadata-store
```

Alternatively, a different namespace can be used to restore Supply Chain Security Tools – Store. In this case, Supply Chain Security Tools – Store API is not available due to conflicting definitions in the RBAC proxy configuration, causing all requests to fail with an `Unauthorized` error. In this scenario, the postgres instance is still accessible, and tools such as `pg_dump` can be used to retrieve table contents and restore in a new live installation of Supply Chain Security Tools – Store.

```
velero restore create restore-metadata-store-$timestamp --from-backup metadata-store
-$timestamp --namespace-mappings metadata-store:restored-metadata-store
```

Currently, mounting an existing `PersistentVolume` or `PersistentVolumeClaim` during installation is not supported.

The minimum suggested resources for backups are `PersistentVolume`, `PersistentVolumeClaim` and `Secret`. The database password `Secret` is needed to set up a Postgres instance with the correct password to properly read data from the restored volume.

Log configuration and usage for Supply Chain Security Tools - Store

This topic describes how you can configure Supply Chain Security Tools (SCST) - Store to output and interpret detailed log information.

Verbosity levels

There are six verbosity levels that Supply Chain Security Tools - Store supports.

Level	Description
Trace	Output extended debugging logs.
Debug	Output standard debugging logs.
More	Output more verbose informational logs.
Default	Output standard informational logs.
Less	Outputs less verbose informational logs.
Minimum	Outputs a minimal set of informational logs.

When SCST - Store is deployed at a specific verbosity level, all logs of that level and lower are output to the console. For example, setting the verbosity level to `More` outputs logs from `Minimal` to `More`, while `Debug` and `Trace` logs are muted.

Currently, the application logs output at these levels:

- **Minimum** does not output any logs.
- **Less** outputs a single log line indicating the current verbosity level is configured in Metadata Store when the application starts.
- **Default** outputs API endpoint access information.
- **Debug** outputs API endpoint payload information, both for requests and responses.
- **Trace** outputs verbose debug information about the actual SQL queries for the database.

Other log levels do not output any additional log information and are present for future extensibility.

If you don't specify a verbosity level when the Store is installed, the level is set to `default`.

Slow SQL

A slow SQL statement is logged only when verbosity level is set to `trace` and the SQL query takes more than 200 milliseconds to execute. You can change this default by setting the `db_slow_threshold_ms` value in values.yaml file and redeploying metadata store.

Error logs

Error logs are always output regardless of the verbosity level, even when set to `minimum`.

Obtaining logs

Kubernetes pods emit logs. The deployment has two pods, one for the database and one for the API back end.

Use `kubectl get pods` to obtain the names of the pods:

```
kubectl get pods -n metadata-store
```

For example:

```
$ kubectl get pods -n metadata-store
NAME                                READY   STATUS    RESTARTS   AGE
metadata-store-app-67659bbc66-2rc6k   2/2     Running   0          4d3h
metadata-store-db-64d5b88587-8dns7    1/1     Running   0          4d3h
```

The database pod has the prefix `metadata-store-db-`. The API backend pod has the prefix `metadata-store-app-`. Use `kubectl logs` to get the logs from the pod you're interested in. For example, to get the logs of the database pod, run:

```
$ kubectl logs metadata-store-db-64d5b88587-8dns7 -n metadata-store
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.
...
```

The API backend pod has two containers, one for `kube-rbac-proxy`, and one for the API server. Use the `--all-containers` flag to see logs from both containers. For example:

```
$ kubectl logs metadata-store-app-67659bbc66-2rc6k --all-containers -n metadata-store
I1206 18:34:17.686135       1 main.go:150] Reading config file: /etc/kube-rbac-proxy/c
```

```
onfig-file.yaml
I1206 18:34:17.784900       1 main.go:180] Valid token audiences:
...
{"level":"info","ts":"2022-05-27T13:47:52.54099339Z","logger":"MetadataStore","msg":"L
og settings","hostname":"metadata-store-app-5c9d6bccdb-kcrt2","LOG_LEVEL":"default"}
{"level":"info","ts":"2022-05-27T13:47:52.5411336992","logger":"MetadataStore","ms
g":"Server Settings","hostname":"metadata-store-app-5c9d6bccdb-kcrt2","bindingaddres
s":"localhost:9443"}
{"level":"info","ts":"2022-05-27T13:47:52.541150096Z","logger":"MetadataStore","ms
g":"Database Settings","hostname":"metadata-store-app-5c9d6bccdb-kcrt2","maxopenconnec
tion":10,"maxidleconnection":100,"connectionmaxlifetime":60}
```

Note

The `kube-rbac-proxy` container uses a different log format than the Store. For information about the proxy's container log format, see Logging Formats in GithHub.

API endpoint log output

When an API endpoint handles a request, the Store generates two and five log entries:

- When the endpoint receives a request, it outputs a `Processing request` line. This logline is shown at the `default` verbosity level.
- If the endpoint includes query or path parameters, it outputs a `Request parameters` line. This line logs the parameters passed in the request. This line is shown at the `default` verbosity level.
- If the endpoint takes in a request body, it outputs a `Request body` line. This line outputs the entire request body as a string. This line is shown at the `debug` verbosity level.
- When the endpoint returns a response, it outputs a `Request response` line. This line is shown at the `default` verbosity level.
- If the endpoint returns a response body, it outputs a second `Request response` line with an extra key `payload`, and its value is set to the entire response body. This line is shown at the `debug` verbosity level.

Format

The logs use JSON output format.

When the Store handles a request, it outputs API endpoint access information in the following format:

```
{"level":"info","ts":"2022-05-27T15:41:36.0519917492","logger":"MetadataStore","ms
g":"Processing request","hostname":"metadata-store-app-c7c8648f7-8dmdl","method":"GE
T","endpoint":"/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8
495f851d0f4bf57b0bab6"}
```

Key-value pairs

Because JSON output format uses key-value pairs, the tables in the following sections list each key and the meaning of their values.

Common to all logs

The following key-value pairs are common for all logs.

Key	Type	Verbosity Level	Description
level	string	all	The log level of the message. This is either `error` for error messages, or `info` for all other messages.
ts	string	all	The timestamp when the log entry was generated. It uses RFC 3339 format with nanosecond precision and 00:00 offset from UTC, meaning Zulu time.
logger	string	all	Used to identify what produced the log entry. For Store, the name always starts with `MetadataStore`. For log entries that display the raw SQL queries, the name is `MetadataStore.gorm`
msg	string	all	A short description of the logged event
hostname	string	all	The Kubernetes host name of the pod handling the request. This helps identify the specific instance of the Store when you deploy multiple instances on a cluster.
error	string	all	The error message which is only available in error log entries
endpoint	string	default	The API endpoint the Metadata Store attempts to handle the request. This also includes any query and path parameters passed in.
method	string	default	The HTTP verb to access the endpoint. For example, `GET` or `POST`.
code	integer	default	The HTTP response code
response	string	default	The HTTP response in human-readable format. For example, `OK`, `Bad Request`, or `Internal Server Error`.
function	string	debug	The function name that handles the request.

Logging query and path parameter values

Those endpoints that use query or path parameters are logged on the `Request parameters` log entry as key-value pairs. They are appended to all other log entries of the same request as key-value pairs.

The key names are the query or path parameter's name, while the value is set to the value of those parameters in string format.

For example, the following log entry contains the `digest` and `id` key, which represents the respective `digest` and `id` query parameters, and their values:

```
{"level":"info","ts":"2022-05-27T15:41:36.052063176Z","logger":"MetadataStore","msg":"Request parameters","hostname":"metadata-store-app-c7c8648f7-8dmdl","method":"GET","endpoint":"/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6","id":0,"digest":"sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6","name":""}
```

These key-value pairs show up in all subsequent log entries of the same call. For example:

```
{"level":"info","ts":"2022-05-27T15:41:36.057393519Z","logger":"MetadataStore","msg":"Request response","hostname":"metadata-store-app-c7c8648f7-8dmdl","method":"GET","endpoint":"/api/images?digest=sha256%3A20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6","id":0,"digest":"sha256:20521f76ff3d27f436e03dc666cc97a511bbe71e8e8495f851d0f4bf57b0bab6","name":"","code":200,"response":"OK"}
```

This is done to:

- Ensure that the application interprets the values of the query or path parameters correctly.

- Figure out which log entries are associated with a particular API request. Because there might be several simultaneous endpoint calls, this is a first attempt at grouping logs by specific calls.

API payload log output

As mentioned at the start of this section, by setting the verbosity level to `debug`, the Store logs the body payload data for both the request and response of an API call.

The `debug` verbosity level, instead of the `default`, displays this information instead of `default` because:

- Body payloads can contain full CycloneDX and SBOM information. Moving the payload information at this level helps keep the production log output to a reasonable size.
- Some information in these payloads might be sensitive, and the user might not want them exposed in production environment logs.

GraphQL endpoint log output

Note

This section is only applicable to Artifactory Metadata Repository (AMR) logs.

When an GraphQL endpoint handles a request, the AMR generates following types of logs:

- Every request received produces a `Processing request` log, which includes the name of the operation called and the requested fields.
- Every response will produces a log containing the actual query and the return status
- If the endpoint returns a response body, it outputs a second `Request response` line with an extra key `payload`, and its value is set to the entire response body. This line is shown at the `debug` verbosity level

Format

The logs output are in JSON format.

When the AMR handles a request, it outputs some GraphQL endpoint access information in the following format:


```
{"level":"info","ts":"2023-03-23T13:11:31.161531-06:00","logger":"Artifact Metadata Re
pository","msg":"Processing request","hostname":"xyzp2DMD6R.vmware.com","getAllApp
s":"query getAllApps {\n  apps(latest:true) {\n    \n    timestamp\n    location {\n
alias\n    }\n  }\n}"}
{"level":"info","ts":"2023-03-23T13:11:31.172953-06:00","logger":"Artifact Metadata Re
pository","msg":"Request response","hostname":"xyzp2DMD6R.vmware.com","getAllApps":"qu
ery getAllApps {\n  apps(latest:true) {\n    \n    timestamp\n    location {\n      al
ias\n    }\n  }\n}","code":200,"response":"OK"}
```


Key-value pairs

The following tables list the meaning of each key found in the logs.

Common to all logs

The following key-value pairs are common for all logs.

Key	Type	Verbosity Level	Description
level	string	all	The log level of the message. This is either `error` for error messages, or `info` for all other messages.
ts	string	all	The timestamp when the log entry was generated. It uses RFC 3339 format with nanosecond precision and 00:00 offset from UTC, meaning Zulu time.
logger	string	all	Used to identify what produced the log entry. For Store, the name always starts with `MetadataStore`. For log entries that display the raw SQL queries, the name is `MetadataStore.gorm`.
msg	string	all	A short description of the logged event
hostname	string	all	The Kubernetes host name of the pod handling the request. This helps identify the specific instance of the Store when you deploy multiple instances on a cluster.
error	string	all	The error message which is only available in error log entries
code	integer	default	The HTTP response code
response	string	default	The HTTP response in human-readable format. For example, `OK`, `Bad Request`, or `Internal Server Error`.
query	string	debug	The operation name is the key and value fields that the fields requested.

API payload log output

By setting the verbosity level to `debug`, the AMR logs the body payload data for both the request and response of an API call.

The `debug` verbosity level, instead of the `default`, displays this information instead of `default` because body payloads can be large and some information in these payloads might be sensitive. You might not want the sensitive payloads exposed in production environment logs.

Logs containing payload information might be in the following format:

```
{"level":"info","ts":"2023-03-23T13:11:31.172966-06:00","logger":"Artifact Metadata Repository","msg":"Request response","hostname":"xyzp2DMD6R.vmware.com","getAllApps":"query getAllApps {\n  apps(latest:true) {\n    \n    timestamp\n    location {\n      alias\n    }\n  }\n}","payload":{"apps":[{"timestamp":"2023-03-22T15:09:38.867371-06:00","location":{"alias":"1-Alias"}}]}}
```

Slow SQL query log output

When the verbosity level is set to `trace`, you see log entries containing slow SQL queries.

Note

Some information in these SQL Query `trace` logs might be sensitive, and you might not want them exposed in production environment logs.

SQL Query log output

Slow SQL query logs are displayed in the following format when verbosity level is set to `trace`:

```
{"level":"info","ts":"2023-03-23T12:48:12.337749-06:00","logger":"Artifact Metadata Repository.gorm","msg":"slow sql >= 200ms","hostname":"xyzp2DMD6R.vmware.com","rows":50000,"sql":"SELECT \"artifact_apps\".\"id\",\"artifact_apps\".\"created_at\",\"artifact_
```

```
apps\".\"updated_at\",\"artifact_apps\".\"deleted_at\",\"artifact_apps\".\"location_id
\",\"artifact_apps\".\"correlation_id\",\"artifact_apps\".\"image_url\",\"artifact_app
s\".\"image_digest\",\"artifact_apps\".\"namespace\",\"artifact_apps\".\"name\",\"arti
fact_apps\".\"instances\",\"artifact_apps\".\"status\",\"artifact_apps\".\"timestamp\"
FROM \"artifact_apps\" INNER JOIN (select max(timestamp) as timestamp, name, namespac
e, location_id from artifact_apps group by location_id, name, namespace) as argo on ar
go.timestamp = artifact_apps.timestamp and argo.name = artifact_apps.name and argo.loc
ation_id = artifact_apps.location_id and argo.namespace = artifact_apps.namespace WHER
E \"artifact_apps\".\"deleted_at\" IS NULL"}
```


It is similar to the API endpoint log output format, but uses the following key-value pairs:

Key	Type	Log Level	Description
rows	integer	trace	Indicates the number of rows affected by the SQL query
sql	string	trace	Displays the raw SQL query for the database
data#	string	all	Used in error log entries. You can replace `#` with an integer because multiples of these keys can appear in the same log entry. These keys contain extra information related to the error.

SQL Query log output

Some Store logs display the executed SQL query commands when you set the verbosity level to `trace` or a SQL call fails.

Note

Some information in these SQL Query trace logs might be sensitive, and you might not want them exposed in production environment logs.

Format

When the Store display SQL query logs, it uses the following format:


```
{"level":"info","ts":"2022-05-27T15:37:26.186960324Z","logger":"MetadataStore.gorm","m
sg":"sql call","hostname":"metadata-store-app-c7c8648f7-8dmdl","rows":1,"sql":"SELECT
count(*) FROM information_schema.tables WHERE table_schema = CURRENT_SCHEMA() AND tabl
e_name = 'images' AND table_type = 'BASE TABLE'"}
```


It is similar to the API endpoint log output format, but uses the following key-value pairs:

Key	Type	Log Level	Description
rows	integer	trace	Indicates the number of rows affected by the SQL query
sql	string	trace	Displays the raw SQL query for the database
data#	string	all	Used in error log entries. You can replace `#` with an integer because multiples of these keys can appear in the same log entry. These keys contain extra information related to the error.

Connect to the PostgreSQL database

You can use a PostgreSQL database with Supply Chain Security Tools - Store. To connect to the PostgreSQL database, you need the following values:

- database name

- user name
- password
- database host
- database port
- database CA certificate

Connect to the PostgreSQL database:

1. Obtain the database name, user name, password, and CA certificate. Run:

   ```
   db_name=$(kubectl get secret postgres-db-secret -n metadata-store -o json | jq
   -r '.data.POSTGRES_DB' | base64 -d)
   db_username=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
   jq -r '.data.POSTGRES_USER' | base64 -d)
   db_password=$(kubectl get secret postgres-db-secret -n metadata-store -o json |
   jq -r '.data.POSTGRES_PASSWORD' | base64 -d)

   db_ca_dir=$(mktemp -d -t ca-cert-XXXX)
   db_ca_path="$db_ca_dir/ca.crt"
   kubectl get secrets postgres-db-tls-cert -n metadata-store -o json | jq -r '.da
   ta."ca.crt"' | base64 -d > $db_ca_path
   ```

   If the password was auto-generated, the `password` command returns an empty string. Run:

   ```
   db_password=$(kubectl get secret postgres-db-password -n metadata-store -o json
   | jq -r '.data.DB_PASSWORD' | base64 -d)
   ```

2. In a separate terminal, run:

   ```
   kubectl port-forward service/metadata-store-db 5432:5432 -n metadata-store
   ```

3. Set the database host and port values on the first terminal:

   ```
   db_host="localhost"
   db_port=5432
   ```

4. To port forward to a different local port number, use the following command template:

   ```
   kubectl port-forward service/metadata-store-db <LOCAL_PORT>:5432 -n metadata-st
   ore
   ```

   Where `LOCAL-PORT` is the port number for the database you want to use.

You can now connect to the database and make queries. For example:

```
psql "host=$db_host port=$db_port user=$db_username dbname=$db_name sslmode=verify-ca
sslrootcert=$db_ca_path" -c "SELECT * FROM images"
```

You can use GUI clients such as Postico or DBeaver to interact with the database.

Troubleshooting Supply Chain Security Tools - Store

This topic contains ways you can troubleshoot known issues for Supply Chain Security Tools (SCST) - Store.

Querying by `insight source` returns zero CVEs even though there are CVEs in the source scan

Symptom

The `insight source get` and other `insight source` commands return zero results.

Solution

You might have to include different combinations of `--repo`, `--org`, `--commit` due to how the scan-controller populates the software bill of materials (SBOM). For more information see Query vulnerabilities, images, and packages.

Persistent volume retains data

Symptom

If **Supply Chain Security Tools - Store** is deployed, deleted, redeployed, and the database password is changed during the redeployment, the `metadata-store-db` pod fails to start. This is caused by the persistent volume used by postgres retaining old data, even though the retention policy is set to `DELETE`.

Solution

Caution

Changing the database password deletes your Supply Chain Security Tools - Store data.

To redeploy the app, either use the same database password or follow these steps to erase the data on the volume:

1. Deploy metadata-store app by using `kapp`.
2. Verify that the `metadata-store-db-*` pod fails.
3. Run:

   ```
   kubectl exec -it metadata-store-db-<some-id> -n metadata-store /bin/bash
   ```

   Where `<some-id>` is the ID generated by Kubernetes and appended to the pod name.
4. Run `rm -rf /var/lib/postgresql/data/*` to delete all database data.

   Where `/var/lib/postgresql/data/*` is the path found in `postgres-db-deployment.yaml`.
5. Delete the `metadata-store` app by using `kapp`.
6. Deploy the `metadata-store` app by using `kapp`.

Missing persistent volume

Symptom

After SCST - Store is deployed, `metadata-store-db` pod might fail for missing volume while `postgres-db-pv-claim` pvc is in `PENDING` state.

This is because the cluster where SCST - Store is deployed does not have `storageclass` defined. `storageclass`'s provisioner is responsible for creating the persistent volume after `metadata-store-db` attaches `postgres-db-pv-claim`.

Solution

1. Verify that your cluster has `storageclass` by running `kubectl get storageclass`.
2. Create a `storageclass` in your cluster before deploying SCST - Store. For example:

```
# This is the storageclass that Kind uses
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provision
er/master/deploy/local-path-storage.yaml

# set the storage class as default
kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storage
class.kubernetes.io/is-default-class":"true"}}}'
```

Builds fail due to volume errors on EKS running Kubernetes v1.23

Symptom

When installing SCST - Store on or upgrading an existing EKS cluster to Kubernetes v1.23, the satabase pod shows:

```
running PreBind plugin "VolumeBinding": binding volumes: provisioning failed for PVC
"postgres-db-pv-claim"
```

Explanation

This is due to the CSIMigrationAWS in this Kubernetes version which requires users to install the Amazon Elastic Block Store (EBS) CSI Driver to use EBS volumes.

SCST - Store uses the default storage class which uses EBS volumes by default on EKS.

Solution

Follow the AWS documentation to install the Amazon EBS CSI Driver before installing SCST - Store or before upgrading to Kubernetes v1.23.

Certificate Expiries

Symptom

The Insight CLI or the Scan Controller fails to connect to SCST - Store.

The logs of the metadata-store-app pod show the following error:

```
$ kubectl logs deployment/metadata-store-app -c metadata-store-app -n metadata-store
...
2022/09/12 21:22:07 http: TLS handshake error from 127.0.0.1:35678: write tcp 127.0.0.
1:9443->127.0.0.1:35678: write: broken pipe
...
```

or

The logs of metadata-store-db show the following error:

```
$ kubectl logs statefulset/metadata-store-db -n metadata-store
...
2022-07-20 20:02:51.206 UTC [1] LOG:  database system is ready to accept connections
2022-09-19 18:05:26.576 UTC [13097] LOG:  could not accept SSL connection: sslv3 alert
bad certificate
...
```

Explanation

cert-manager rotates the certificates, but the metadata-store and the PostgreSQL db are unaware of the change, and are using the old certificates.

Solution

If you see `TLS handshake error` in the metadata-store-app logs, delete the metadata-store-app pod and wait for it to come back up.

```
kubectl delete pod metadata-store-app-xxxx -n metadata-store
```

If you see `could not accept SSL connection` in the metadata-store-db logs, delete the metadata-store-db pod and wait for it to come back up.

```
kubectl delete pod metadata-store-db-0 -n metadata-store
```

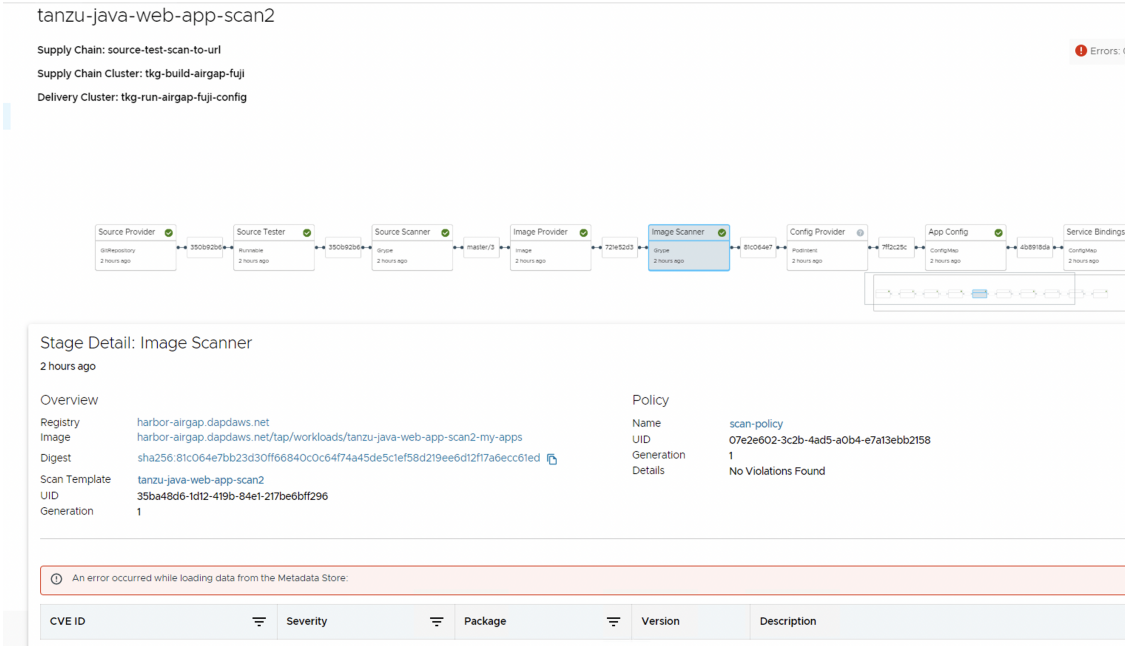
Troubleshooting errors from Tanzu Application Platform GUI related to SCST - Store

Different Tanzu Application Platform GUI plug-ins use SCST - Store to display information about vulnerabilities and packages. Some errors visible in Tanzu Application Platform GUI are related to this connection.

An error occurred while loading data from the Metadata Store

Symptom

In the Supply Chain Choreographer plug-in, you see the error message `An error occurred while loading data from the Metadata Store`.



Cause

There are multiple potential causes. The most common cause is `tap-values.yaml` missing the configuration that enables Tanzu Application Platform GUI to communicate with Supply Chain Security Tools - Store.

Solution

See Supply Chain Choreographer - Enable CVE scan results for the necessary configuration to add to `tap-values.yaml`. After adding the configuration, update your Tanzu Application Platform deployment or Tanzu Application Platform GUI deployment with the new values.

Troubleshoot upgrading Supply Chain Security Tools - Store

This topic describes how you can troubleshoot upgrading issues Supply Chain Security Tools (SCST) - Store.

Database deployment does not exist

To prevent issues with the metadata store database, such as the ones described in this topic, the database deployment is `StatefulSet` in

- Tanzu Application Platform v1.2 and later
- Metadata Store v1.1 and later

If you have scripts searching for a `metadata-store-db` deployment, edit the scripts to instead search for `StatefulSet`.

Invalid checkpoint record

When using Tanzu to upgrade to a new version of the store, there is occasionally data corruption. Here is an example of how this shows up in the log:

```
PostgreSQL Database directory appears to contain a database; Skipping initialization

2022-01-21 21:53:38.799 UTC [1] LOG:  starting PostgreSQL 13.5 (Ubuntu 13.5-1.pgdg18.0
4+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0, 64-b
it
2022-01-21 21:53:38.799 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2022-01-21 21:53:38.799 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2022-01-21 21:53:38.802 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.
s.PGSQL.5432"
2022-01-21 21:53:38.807 UTC [14] LOG:  database system was shut down at 2022-01-21 21:
21:12 UTC
2022-01-21 21:53:38.807 UTC [14] LOG:  invalid record length at 0/1898BE8: wanted 24,
got 0
2022-01-21 21:53:38.807 UTC [14] LOG:  invalid primary checkpoint record
2022-01-21 21:53:38.807 UTC [14] PANIC:  could not locate a valid checkpoint record
2022-01-21 21:53:39.496 UTC [1] LOG:  startup process (PID 14) was terminated by signa
l 6: Aborted
2022-01-21 21:53:39.496 UTC [1] LOG:  aborting startup due to startup process failure
2022-01-21 21:53:39.507 UTC [1] LOG:  database system is shut down
```

The log shows a database pod in a failure loop. For steps to fix the issue so that the upgrade can proceed, see the SysOpsPro documentation.

Upgraded pod hanging

Because the default access mode in the PVC is `ReadWriteOnce`, if you are deploying in an environment with multiple nodes then each pod might be on a different node. This causes the upgraded pod to spin up but then get stuck initializing because the original pod does not stop. To

resolve this issue, find and delete the original pod so that the new pod can attach to the persistent volume:

1. Discover the name of the app pod that is not in a pending state by running:

   ```
   kubectl get pods -n metadata-store
   ```

2. Delete the pod by running:

   ```
   kubectl delete pod METADATA-STORE-APP-POD-NAME -n metadata-store
   ```

Failover, redundancy, and backups for Supply Chain Security Tools - Store

This topic describes how you can configure and use failover, redundancy, and backups for Supply Chain Security Tools (SCST) - Store.

API Server

By default the API server has 1 replica. If the pod fails, the single instance restarts by normal Kubernetes behavior, but there is downtime. If you upgrade, some downtime is possible.

You can configure the number of replicas by using the `app_replicas` text box in the `scst-store-values.yaml` file.

Database

By default, the database has one replica, and restarts with some downtime if it fails.

> **Caution**
>
> Although you can configure `db_replicas` in `scst-store-values.yaml`, this is discouraged because `db_replicas` is still experimental. The default internal database is not for production use. For production deployments, use an external database.

- Use external postgres database
- AWS RDS postgres configuration

For the default PostgreSQL database deployment, with `deploy_internal_db` set to `true`, you can use `Velero` as the backup method. For information about using `Velero` as the backup method, see Backups.

Custom certificate configuration for Supply Chain Security Tools - Store

This topic describes how you can configure the following certificates for Supply Chain Security Tools (SCST) - Store:

1. Default configuration
2. Custom certificate

Default configuration

By default, SCST - Store creates a self-signed certificate and TLS communication is automatically enabled.

If ingress support is enabled, SCST - Store installation creates an HTTPProxy entry with host routing by using the qualified name `metadata-store.<ingress_domain>`. For example, `metadata-store.example.com`. The created route supports HTTPS communication using the self-signed certificate with the same subject `Alternative Name`.

(Optional) Setting up custom ingress TLS certificate

(Optional) Users can configure TLS to use a custom certificate. To do that:

1. Place the certificates in the secret.
2. Edit the `tap-values.yaml` to use this secret.

Place the certificates in secret

1. Create the certificate secret before deploying SCST - Store.
2. Create a Kubernetes object with kind `Secret` and type `kubernetes.io/tls`.

Update `tap-values.yaml`

1. In the `tap-values.yaml` file, you can configure the metadata store to use the `namespace` and `secretName` from the secret created in the last step.

```
metadata_store:
  tls:
    namespace: "namespace"
    secretName: "secretName"
```

Where:

- `namespace` is the targeted namespace for secret consumption by the HTTPProxy.
- `secretName` is the name of secret for consumption by the HTTPProxy.

Additional resources

- Ingress support
- TLS configuration

TLS configuration for Supply Chain Security Tools - Store

This topic describes how you can configure TLS for Supply Chain Security Tools (SCST) - Store.

Important

SCST - Store only supports TLS v1.2.

Setting up custom ingress TLS ciphers

In the `tap-values.yaml` file, `tls.server.rfcCiphers` are set as shown in the following YAML:

```
metadata_store:
  tls:
    server:
```

```
      rfcCiphers:
        - TLS_AES_128_GCM_SHA256
        - TLS_AES_256_GCM_SHA384
        - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

Where `tls.server.rfcCiphers` is a list of cipher suites for the server. Values are from the Go TLS package constants. If you omit values, the default Go cipher suites are used. These are the default values:

- `TLS_AES_128_GCM_SHA256`
- `TLS_AES_256_GCM_SHA384`
- `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`

Example custom TLS settings

The following is a complete example of TLS configuration:

```
metadata_store:
  tls:
    namespace: NAMESPACE
    secretName: SECRET-NAME
    server:
      rfcCiphers:
        - TLS_AES_128_GCM_SHA256
        - TLS_AES_256_GCM_SHA384
        - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

Where:

- `NAMESPACE` is the name of the namespace you want to configure TLS with.
- `SECRET-NAME` is the name of the secret you want to configure TLS with.

Additional resources

- Custom certificate configuration
- Ingress support

Certificate rotation for Supply Chain Security Tools - Store

This topic describes how you can rotate TLS certificates for Supply Chain Security Tools (SCST) - Store.

Certificates

By default, the `use_cert_manager` setting is set to `"true"`. When the setting `use_cert_manager` is `"true"` the Store uses `cert-manager` to generate a CA certificate, an API certificate, and a database

Certificate.

To see these certificates:

```
$ kubectl get certificate -n metadata-store
NAME                     READY   SECRET                   AGE
app-tls-ca-cert          True    app-tls-ca-cert          38d
app-tls-cert             True    app-tls-cert             38d
postgres-db-tls-cert     True    postgres-db-tls-cert     38d
```

The earlier certificates are automatically rotated by `cert-manager`.

The Store can run these certificates automatically once `cert-manager` rotates them.

If the environment is a multi-cluster setup, the operator must manually copy over the new CA certificate to the build cluster.

Certificate duration setting

In the `tap-values.yaml` file, `api_cert_duration`, `api_cert_renew_before`, `ca_cert_duration`, and `ca_cert_renew_before` are configurable as shown in the following YAML:

```
metadata_store:
  ca_cert_duration: CA-DURATION
  ca_cert_renew_before: CA-RENEW
  api_cert_duration: API-DURATION
  api_cert_renew_before: API-RENEW
```

Where:

- `CA-DURATION` is the duration that the ca certificate is valid for. Must be given in `h`, `m`, or `s`. Default value is 8760h.
- `CA-RENEW` is how long before the expiry of the ca certificate is renewed. Must be given in `h`, `m`, or `s`. Default value is 1h.
- `API-DURATION` is the duration that the API certificate is valid for. Must be given in `h`, `m`, or `s`. Default value is 2160h.
- `API-RENEW` is how long before the expiry of the API certificate is renewed. Must be given in h, m, or s. Default value is 24h.

Important

The `*_cert_duration` and the corresponding `*_renew_before` settings must not be close. For more information, see the cert-manager documentation. This can lead to a renewal loop. The `*_cert_duration` must be greater than the corresponding `*_renew_before`. The earlier settings only take effect when `use_cert_manager` is `"true"`. If the `use_cert_manager` is not set, it defaults to `"true"`.

Ingress support for Supply Chain Security Tools - Store

This topic describes how to configure ingress for Supply Chain Security Tools (SCST) - Store.

Ingress configuration

Supply Chain Security Tools (SCST) - Store has ingress support by using Contour's HTTPProxy resources. To enable ingress support, a Contour installation must be available in the cluster.

To change ingress configuration, edit your `tap-values.yaml` when you install a Tanzu Application Platform profile. When you configure the `shared.ingress_domain` property, SCST - Store automatically uses that setting.

Alternatively, you can customize SCST - Store's configuration under the `metadata_store` property. Under `metadata_store`, there are two values to configure the proxy:

- `ingress_enabled`
- `ingress_domain`

This is an example snippet in a `tap-values.yaml`:

```
...
metadata_store:
  ingress_enabled: "true"
  ingress_domain: "example.com"
  app_service_type: "ClusterIP"  # Defaults to `LoadBalancer`. If ingress is enabled t
hen this must be set to `ClusterIP`.
...
```

SCST - Store installation creates an HTTPProxy entry with host routing by using the qualified name `METADATA-STORE.INGRESS-DOMAIN`. For example, `metadata-store.example.com`. The route supports HTTPS communication using a certificate. By default, a self-signed certificate is used with the same subject `alternative name`. For more information, see Custom certificate configuration.

Contour and DNS setup are not part of SCST - Store installation. Access to SCST - Store using Contour depends on the correct configuration of these two components.

Make the proper DNS record available to clients to resolve `metadata-store` and set `ingress_domain` to Envoy service's external IP address.

DNS setup example:

```
$ kubectl describe svc envoy -n tanzu-system-ingress
> ...
  Type:                     LoadBalancer
  ...
  LoadBalancer Ingress:     100.2.3.4
  ...
  Port:                     https  443/TCP
  ...

$ nslookup metadata-store.example.com
> Server:    8.8.8.8
  Address:  8.8.8.8#53

  Non-authoritative answer:
  Name:  metadata-store.example.com
  Address: 100.2.3.4

$ curl https://metadata-store.example.com/api/health -k -v
> ...
  < HTTP/2 200
  ...
```

Note

The preceding `curl` example uses the not secure `-k` flag to skip TLS verification because the Store installs a self-signed certificate. The following section shows how to access the CA certificate to enable TLS verification for HTTP clients.

Get the TLS CA certificate

To get SCST - Store's TLS CA certificate, use `kubectl get secret`. In this example, you save the certificate for the environment variable to a file.

```
kubectl get secret CERT-NAME -n metadata-store -o json | jq -r '.data."ca.crt"' | base
64 -d > OUTPUT-FILE
```

Where:

- `CERT-NAME` is the name of the certificate. This must be `ingress-cert` if no custom certificate is used.
- `OUTPUT-FILE` is the file you want to create to store the certificate.

For example:

```
$ kubectl get secret tap-ingress-selfsigned-root-ca -n cert-manager -o json | jq -r '.
data."ca.crt"' | base64 -d > insight-ca.crt
$ cat insight-ca.crt
```

Additional Resources

- Custom certificate configuration
- TLS configuration
- Certificate rotation
- Configure target endpoint and certificate

Use your LoadBalancer with Supply Chain Security Tools - Store

This topic describes how to use your LoadBalancer with Supply Chain Security Tools (SCST) - Store.

Configure LoadBalancer

Note

`LoadBalancer` is not the recommended service type. Consider the recommended configuration of enabling Ingress.

To configure a `LoadBalancer`:

1. Edit `/etc/hosts/` to use the external IP address of the `metadata-store-app` service.

   ```
   METADATA_STORE_IP=$(kubectl get service/metadata-store-app --namespace metadata
   -store -o jsonpath="{.status.loadBalancer.ingress[0].ip}")
   METADATA_STORE_PORT=$(kubectl get service/metadata-store-app --namespace metada
   ta-store -o jsonpath="{.spec.ports[0].port}")
   METADATA_STORE_DOMAIN="metadata-store-app.metadata-store.svc.cluster.local"

   # Delete any previously added entry
   sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

   echo "$METADATA_STORE_IP $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /de
   v/null
   ```

> **Note**
>
> On EKS, you must get the IP address for the LoadBalancer. Find the IP address by running something similar to the following: `dig RANDOM-SHA.us-east-2.elb.amazonaws.com`. Where `RANDOM-SHA` is the EXTERNAL-IP received for the LoadBalancer.

2. Select one of the IP addresses returned from the `dig` command and write it to the `/etc/hosts` file.

Port forwarding

If you want to use port forwarding instead of the external IP address from the `LoadBalancer`, follow these steps:

Configure port forwarding for the service so the insight plug-in can access SCST - Store. Run:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

Note: You must run the port forwarding command in a separate terminal window, or run the command in the background: `kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store &`

Edit your `/etc/hosts` file for Port Forwarding

Use the following script to add a new local entry to `/etc/hosts`:

```
METADATA_STORE_PORT=$(kubectl get service/metadata-store-app --namespace metadata-stor
e -o jsonpath="{.spec.ports[0].port}")
METADATA_STORE_DOMAIN="metadata-store-app.metadata-store.svc.cluster.local"

# delete any previously added entry
sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "127.0.0.1 $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

Configure the Insight plug-in

Because you deployed Supply Chain Security Tools (SCST) - Store without using Ingress, you must use the Certificate resource `app-tls-cert` for HTTPS communication.

To get the CA Certificate:

```
kubectl get secret app-tls-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | b
ase64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --
ca-cert insight-ca.crt
```

> **Important**
>
> The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA

certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to add data and query data.

Use your NodePort with Supply Chain Security Tools - Store

This topic describes how you can use your NodePort with Supply Chain Security Tools (SCST) - Store.

Overview

Note

The recommended service type is Ingress. NodePort is only recommended when the cluster does not support Ingress or the cluster does not support the LoadBalancer service type. `NodePort` is not supported for a multicluster setup, as certificates cannot be modified.

You must use port forwarding when using the `NodePort` configuration.

Configure port forwarding for the service so the insight plug-in can access SCST - Store. Run:

```
kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store
```

Note: You must run the port forwarding command in a separate terminal window, or run the command in the background: `kubectl port-forward service/metadata-store-app 8443:8443 -n metadata-store &`

Edit your `/etc/hosts` file for Port Forwarding

Use the following script to add a new local entry to `/etc/hosts`:

```
METADATA_STORE_PORT=$(kubectl get service/metadata-store-app --namespace metadata-stor
e -o jsonpath="{.spec.ports[0].port}")
METADATA_STORE_DOMAIN="metadata-store-app.metadata-store.svc.cluster.local"

# delete any previously added entry
sudo sed -i '' "/$METADATA_STORE_DOMAIN/d" /etc/hosts

echo "127.0.0.1 $METADATA_STORE_DOMAIN" | sudo tee -a /etc/hosts > /dev/null
```

Configure the Insight plug-in

Because you deployed Supply Chain Security Tools (SCST) - Store without using Ingress, you must use the Certificate resource `app-tls-cert` for HTTPS communication.

To get the CA Certificate:

```
kubectl get secret app-tls-cert -n metadata-store -o json | jq -r '.data."ca.crt"' | b
ase64 -d > insight-ca.crt
```

Set the target by running:

```
tanzu insight config set-target https://$METADATA_STORE_DOMAIN:$METADATA_STORE_PORT --
```

```
ca-cert insight-ca.crt
```

Important

The `tanzu insight config set-target` does not initiate a test connection. Use `tanzu insight health` to test connecting using the configured endpoint and CA certificate. Neither commands test whether the access token is correct. For that you must use the plug-in to add data and query data.

Multicluster setup for Supply Chain Security Tools - Store

This topic describes how you can deploy Supply Chain Security Tools (SCST) - Store in a multicluster setup, including installing multiple profiles such as, View, Build, Run, and Iterate.

Overview

SCST - Store is deployed with the View profile. After installing the View profile, but before installing the Build profile, you must add configuration for SCST - Store to the Kubernetes cluster where you intend to install the Build profile. This topic explains how to add configuration which allows components in the Build cluster to communicate with SCST - Store in the View cluster.

Note

If you already deployed the Build profile, you can follow this procedure. However, in the Install Build profile step, instead of deploying the Build profile again, update your deployment using `tanzu package installed update`.

Prerequisites

You must install the View profile. See Install View profile.

Procedure summary

1. Copy SCST - Store CA certificate from the View cluster.
2. Copy SCST - Store authentication token from the View cluster.
3. Apply the CA certificate and authentication token to the Kubernetes cluster where you intend to install the Build profile.
4. Install the Build profile.

Copy SCST - Store CA certificate from View cluster

With your kubectl targeted at the View cluster, you can view SCST - Store's TLS CA certificate. Run these commands to copy the CA certificate into a file `store_ca.yaml`.

```
CA_CERT=$(kubectl get secret -n metadata-store CERT-NAME -o json | jq -r ".data.\"ca.c
rt\"")
cat <<EOF > store_ca.yaml
---
apiVersion: v1
kind: Secret
type: Opaque
```

```
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
data:
  ca.crt: $CA_CERT
EOF
```

Where `CERT-NAME` is the name of the certificate you want to reference in `store_ca.yaml`.

For example:

```
$ CA_CERT=$(kubectl get secret -n metadata-store ingress-cert -o json | jq -r ".dat
a.\"ca.crt\"")
$ cat <<EOF > store_ca.yaml
---
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: store-ca-cert
  namespace: metadata-store-secrets
data:
  ca.crt: $CA_CERT
EOF
```

Copy SCST - Store authentication token from the View cluster

Copy the SCST - Store authentication token into an environment variable. You use this environment variable in the next step.

```
AUTH_TOKEN=$(kubectl get secrets metadata-store-read-write-client -n metadata-store -o
jsonpath="{.data.token}" | base64 -d)
```

Apply the CA certificate and authentication token to a new Kubernetes cluster

Before you deploy the Build profile, you must apply the CA certificate and authentication token from the earlier steps. Then the Build profile deployment has access to these values.

To apply the CA certificate and authentication token:

1. With your kubectl targeted at the Build cluster, create a namespace for the CA certificate and authentication token.

   ```
   kubectl create ns metadata-store-secrets
   ```

2. Apply the CA certificate `store_ca.yaml` secret YAML you generated earlier.

   ```
   kubectl apply -f store_ca.yaml
   ```

3. Create a secret to store the access token. This uses the `AUTH_TOKEN` environment variable.

   ```
   kubectl create secret generic store-auth-token \
     --from-literal=auth_token=$AUTH_TOKEN -n metadata-store-secrets
   ```

The cluster now has a CA certificate named `store-ca-cert` and authentication token named `store-auth-token` in the namespace `metadata-store-secrets`.

Install Build profile

If you came to this topic from the Install multicluster Tanzu Application Platform profiles topic after installing the View profile, return to that topic to install the Build profile.

The Build profile `values.yaml` contains configuration that references the secrets in the `metadata-store-secrets` namespace you created in this guide. The names of these secrets are hard coded in the example `values.yaml`.

More information about how Build profile uses the configuration

The secrets you created are used in the Build profile `values.yaml` to configure the Grype scanner which talks to SCST - Store. After performing a vulnerabilities scan, the Grype scanner sends the results to SCST - Store. Here's a snippet of what the configuration might look like.

```
...
grype:
  namespace: "MY-DEV-NAMESPACE" # (Optional) Defaults to default namespace.
  targetImagePullSecret: "TARGET-REGISTRY-CREDENTIALS-SECRET"
  metadataStore:
    url: METADATA-STORE-URL-ON-VIEW-CLUSTER # Url with http / https
    caSecret:
        name: store-ca-cert
        importFromNamespace: metadata-store-secrets # Must match with `ingress-cert.da
ta."ca.crt"` of store on view cluster
    authSecret:
        name: store-auth-token # Must match with valid store token of metadata-store o
n view cluster
        importFromNamespace: metadata-store-secrets
...
```

Where:

- `METADATA-STORE-URL-ON-VIEW-CLUSTER` is the ingress URL of SCST - Store deployed to the View cluster. For example, `https://metadata-store.example.com`. See Ingress support.
- `TARGET-REGISTRY-CREDENTIALS-SECRET` is the name of the secret that contains the credentials to pull an image from the registry for scanning.
- `MY-DEV-NAMESPACE` is the name of the developer namespace. SCST - Scan deploys the ScanTemplates there. This allows the scanning feature to run in this namespace.

Configure developer namespaces

After you finish the entire Tanzu Application Platform installation process, you are ready to configure developer namespaces. To prepare developer namespaces, you must export the secrets you created earlier to those namespaces.

Exporting SCST - Store secrets to a developer namespace in a Tanzu Application Platform multicluster deployment

Export secrets to a developer namespace by creating `SecretExport` resources on the developer namespace. Run the following command to create the `SecretExport` resources. You must have created and populated the `metadata-store-secrets` namespace.

```
cat <<EOF | kubectl apply -f -
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-ca-cert
```

```
  namespace: metadata-store-secrets
spec:
  toNamespaces: [DEV-NAMESPACES]
---
apiVersion: secretgen.carvel.dev/v1alpha1
kind: SecretExport
metadata:
  name: store-auth-token
  namespace: metadata-store-secrets
spec:
  toNamespaces: [DEV-NAMESPACES]
EOF
```

Where `[DEV-NAMESPACES]` is an array of developer namespaces where the secrets are exported.

More detailed description of `metadata` configuration can be found at Cluster Specific Store Configuration

Additional resources

- Ingress support
- Custom certificate configuration

Developer namespace setup for Supply Chain Security Tools - Store

This topic describes how you can set up your developer namespace for Supply Chain Security Tools (SCST) - Store.

Overview

After you finish the entire Tanzu Application Platform installation process, you are ready to configure the developer namespace. When you configure a developer namespace, you must export the Supply Chain Security Tools (SCST) - Store CA certificate and authentication token to the namespace. This enables SCST - Scan to find the credentials to send scan results to SCST - Store.

There are two ways to deploy Tanzu Application Platform:

- Single cluster, which entails using the Tanzu Application Platform values file
- Multicluster, which entails using `SecretExport`

Single cluster - Using the Tanzu Application Platform values file

When deploy the Tanzu Application Platform Full or Build profile, edit the `tap-values.yaml` file you used to deploy Tanzu Application Platform.

```
metadata_store:
  ns_for_export_app_cert: "DEV-NAMESPACE"
```

Where `DEV-NAMESPACE` is the name of the developer namespace.

The `ns_for_export_app_cert` supports one namespace at a time. If you have multiple namespaces you can replace this value with a `"*"`, but this exports the CA to all namespaces. Consider whether this increased visibility presents a risk.

```
metadata_store:
  ns_for_export_app_cert: "*"
```

Update Tanzu Application Platform to apply the changes by running:

```
$ tanzu package installed update tap -f tap-values.yaml -n tap-install
```

Multicluster - Using `SecretExport`

In a multicluster deployment, follow the steps in Multicluster setup. It describes how to create secrets and export secrets to developer namespaces.

Next steps

If you arrived in this topic from Setting up the Out of the Box Supply Chain with testing and scanning, return to that topic and continue with the instructions.

Retrieve access tokens for Supply Chain Security Tools - Store

This topic describes how you can retrieve access tokens for Supply Chain Security Tools (SCST) - Store.

Overview

When you install Tanzu Application Platform, the Supply Chain Security Tools (SCST) - Store deployment automatically includes a read-write service account. This service account is bound to the `metadata-store-read-write` role.

There are two types of SCST - Store service accounts:

1. Read-write service account - full access to the `POST` and `GET` API requests
2. Read-only service account - can only use `GET` API requests

This topic shows how to retrieve the access token for these service accounts.

Retrieving the read-write access token

To retrieve the read-write access token, run:

```
kubectl get secrets metadata-store-read-write-client -n metadata-store -o jsonpath="{.
data.token}" | base64 -d
```

Retrieving the read-only access token

In order retrieve the read-only access token, you must first have a read-only service account. See Create read-only service account.

To retrieve the read-only access token, run:

```
kubectl get secrets metadata-store-read-client -n metadata-store -o jsonpath="{.data.t
oken}" | base64 -d
```

Using an access token

The access token is a Bearer token used in the http request header `Authorization`. For example, `Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjhMV0....`

Additional Resources

- Create service accounts
- Create a service account with a custom cluster role

Retrieve and create service accounts for Supply Chain Security Tools - Store

This topic explains how you can create service accounts for Supply Chain Security Tools (SCST) - Store.

Overview

When you install Tanzu Application Platform, the Supply Chain Security Tools (SCST) - Store deployment automatically includes a read-write service account. This service account is bound to the `metadata-store-read-write` role.

There are two types of SCST - Store service accounts:

1. Read-write service account - full access to the `POST` and `GET` API requests
2. Read-only service account - can only use `GET` API requests

Create read-write service account

When you install Tanzu Application Platform, the SCST - Store deployment automatically includes a read-write service account. This service account is already bound to the `metadata-store-read-write` role.

To create an additional read-write service account, run the following command. The command creates a service account called `metadata-store-read-write-client`:

```
kubectl apply -f - -o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get", "create", "update"]
  apiGroups: [ "metadata-store/v1" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-read-write
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-read-write
subjects:
- kind: ServiceAccount
  name: metadata-store-read-write-client
  namespace: metadata-store
---
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-write-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: "metadata-store.apps.tanzu.vmware.com/service-account"
automountServiceAccountToken: false
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: metadata-store-read-write-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-rule: "upsert after upserting metadata-store.apps.tanzu.vmwar
e.com/service-account"
    kubernetes.io/service-account.name: "metadata-store-read-write-client"
EOF
```

Create a read-only service account

You can create a read-only service account with a default cluster role or with a custom cluster role.

With a default cluster role

During Store installation, the `metadata-store-read-only` cluster role is created by default. This cluster role allows the bound user to have `get` access to all resources. To bind to this cluster role, run the following command:

```
kubectl apply -f - -o yaml << EOF
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metadata-store-read-only
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: metadata-store-read-only
subjects:
- kind: ServiceAccount
  name: metadata-store-read-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: "metadata-store.apps.tanzu.vmware.com/service-account"
automountServiceAccountToken: false
---
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-rule: "upsert after upserting metadata-store.apps.tanzu.vmwar
```

```
e.com/service-account"
    kubernetes.io/service-account.name: "metadata-store-read-client"
EOF
```

With a custom cluster role

If using the default role is not sufficient, see Create a service account with a custom cluster role.

Additional Resources

- Retrieve access tokens
- Create a service account with a custom cluster role

Create a service account with a custom cluster role for Supply Chain Security Tools - Store

This topic describes how you can create a service account with a custom cluster role for Supply Chain Security Tools (SCST)- Store.

Example service account

If you do not want to bind to the default cluster role, create a read-only role in the `metadata-store` namespace with a service account. The following example creates a service account named `metadata-store-read-client`:

```
kubectl apply -f - -o yaml << EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: metadata-store-ro
  namespace: metadata-store
rules:
- resources: ["all"]
  verbs: ["get"]
  apiGroups: [ "metadata-store/v1" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: metadata-store-ro
  namespace: metadata-store
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: metadata-store-ro
subjects:
- kind: ServiceAccount
  name: metadata-store-read-client
  namespace: metadata-store
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-group: "metadata-store.apps.tanzu.vmware.com/service-account"
automountServiceAccountToken: false
---
apiVersion: v1
```

```
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: metadata-store-read-client
  namespace: metadata-store
  annotations:
    kapp.k14s.io/change-rule: "upsert after upserting metadata-store.apps.tanzu.vmwar
e.com/service-account"
    kubernetes.io/service-account.name: "metadata-store-read-client"
EOF
```

Additional Resources

- Retrieve access tokens
- Create service accounts

Install Supply Chain Security Tools - Store independent from Tanzu Application Platform profiles

This topic describes how you can install Supply Chain Security Tools (SCST) - Store from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Supply Chain Security Tools - Store. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing SCST - Store:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install cert-manager on the cluster. See Install cert-manager.
- See Deployment Details and Configuration to review what resources are deployed. For more information, see the overview.
- Create ClusterIssuer

  ```
  kubectl apply -f - <<EOF
  apiVersion: cert-manager.io/v1
    kind: ClusterIssuer
    metadata:
      name: tap-ingress-selfsigned
    spec:
      selfSigned: {}
  EOF
  ```

Install

To install SCST - Store:

1. To use this deployment, the user must have set up the Kubernetes cluster to provision persistent volumes on demand. Ensure that a default storage class is available in your

cluster. Verify whether default storage class is set in your cluster using `kubectl get storageClass`.

```
kubectl get storageClass
```

For example:

```
$ kubectl get storageClass
NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
standard (default)   rancher.io/local-path   Delete          WaitForFirstConsum
er   false                  7s
```

2. List version information for the package using `tanzu package available list`.

```
tanzu package available list metadata-store.apps.tanzu.vmware.com --namespace t
ap-install
```

For example:

```
$ tanzu package available list metadata-store.apps.tanzu.vmware.com --namespace
tap-install
- Retrieving package versions for metadata-store.apps.tanzu.vmware.com...
  NAME                          VERSION       RELEASED-AT
  metadata-store.apps.tanzu.vmware.com  1.0.2
```

3. (Optional) List all the available deployment configuration options.

```
tanzu package available get metadata-store.apps.tanzu.vmware.com/VERSION --valu
es-schema -n tap-install
```

Where `VERSION` is the your package version number.

For example:

```
$ tanzu package available get metadata-store.apps.tanzu.vmware.com/1.0.2 --valu
es-schema -n tap-install
| Retrieving package details for metadata-store.apps.tanzu.vmware.com/1.0.2...
  KEY                             DEFAULT
TYPE     DESCRIPTION
  pg_limit_memory                 4Gi
string   Memory limit for postgres container in metadata-store-db deployment
  tls.namespace
string   The targeted namespace for secret consumption by the HTTPProxy.
  add_default_rw_service_account  true
string   Adds a read-write service account which can be used to obtain access t
oken to use metadata-store CLI
  api_host                        localhost
string   The internal hostname for the metadata api endpoint. This will be used
by the kube-rbac-proxy sidecar.
  app_replicas                    1
integer  The number of replicas for the metadata-store-app
  ingress_domain
string   Domain to be used by the HTTPProxy ingress object. The "metadata-stor
e" subdomain will be prepended to the value provided. For example: "example.co
m" would become "metadata-store.example.com". Required if ingress_enabled is tr
ue.
  kube_rbac_proxy_limit_cpu       250m
string   CPU limit for kube-rbac-proxy container in the metadata-store-app depl
oyment
  pg_limit_cpu                    2Gi
string   CPU limit for postgres container in metadata-store-db deployment
  tls.server.minTLSVersion        VersionTLS12
```

```
string   Minimum TLS version supported. Value must match version names from htt
ps://golang.org/pkg/crypto/tls/#pkg-constants. (default "VersionTLS12")
  db_host                         metadata-store-db
string   The address to the postgres database host that the metadata-store app
uses to connect. The default is set to metadata-store-db which is the postgres
service name. Changing this does not change the postgres service name
  db_replicas                     1
integer  The number of replicas for the metadata-store-db
  pg_req_cpu                      1Gi
string   CPU request for postgres container in metadata-store-db deployment
  priority_class_name
string   If specified, this value is the name of the desired PriorityClass for
the metadata-store-db deployment
  tls.secretName
string   The name of secret for consumption by the HTTPProxy.
  db_ca_certificate
string   This should only be set in the case when 'deploy_internal_db' is 'fals
e'. Set this to the trusted CA Certificate that signed the Postgres DB TLS Cert
ificate
  db_password
string   The database user password. If no value is provided, a 32 character va
lue will be generated.
  db_port                         5432
string   The database port to use. This is the port to use when connecting to t
he database pod.
  app_limit_cpu                   250m
string   CPU limit for metadata-store-app container
  auth_proxy_host                 0.0.0.0
string   The binding ip address of the kube-rbac-proxy sidecar
  db_max_open_conns               10
integer  Sets the maximum number of open database connections from the Metadata
Store to the database.
  db_name                         metadata-store
string   The name of the database to use.
  db_user                         metadata-store-user
string   The database user to create and use for updating and querying. The met
adata postgres section create this user. The metadata api server uses this user
name to connect to the database.
  kube_rbac_proxy_req_memory      128Mi
string   Memory request for kube-rbac-proxy container in the metadata-store-app
deployment
  auth_proxy_port                 8443
integer  The external port address of the of the kube-rbac-proxy sidecar
  db_conn_max_lifetime            60
integer  Sets the maximum amount of time a database connection may be reused in
seconds.
  ingress_enabled                 false
string   Contour is required to be installed to use this flag. When true, this
creates an HTTPProxy object for the metadata-store. If false, then no ingress i
s configured.
  storage_class_name
string   The storage class name of the persistent volume used by Postgres datab
ase for storing data. The default value will use the default class name defined
on the cluster.
  api_port                        9443
integer  The internal port for the metadata app api endpoint. This will be used
by the kube-rbac-proxy sidecar.
  app_service_type                LoadBalancer
string   The type of service to use for the metadata app service. This can be s
et to 'Nodeport', 'ClusterIP' or 'LoadBalancer'.
  db_sslmode                      verify-full
string   Determines the security connection between API server and Postgres dat
abase. This can be set to 'verify-ca' or 'verify-full'
  use_cert_manager                true
string   Cert manager is required to be installed to use this flag. When true,
this creates certificates object to be signed by cert manager for the API serve
```

```
r and Postgres database. If false, the certificate object have to be provided b
y the user.
  app_req_cpu                     100m
string   CPU request for metadata-store-app container
  database_request_storage        10Gi
string   The storage requested of the persistent volume used by Postgres databa
se for storing data.
  deploy_internal_db              true
string   If set to 'true', a postgres deployment will be created. If set to 'fa
lse', db_host and db_port should point to an accessible postgres instance. Post
gres connections require TLS, so the corresponding db_ca_certification must be
provided
  kube_rbac_proxy_req_cpu         100m
string   CPU request for kube-rbac-proxy container in the metadata-store-app de
ployment
  ns_for_export_app_cert          scan-link-system
string   The namespace where the "Supply Chain Security Tools for VMware Tanzu
- Scan" component is installed in. Certain certificates will be exported to tha
t namespace so that scan reports can be posted to the Metadata Store.
  pg_req_memory                   1Gi
string   Memory request for postgres container in metadata-store-db deployment
  app_limit_memory                512Mi
string   Memory limit for metadata-store-app container
  app_req_memory                  128Mi
string   Memory request for metadata-store-app container
  db_max_idle_conns               100
integer  Sets the maximum number of database connections from the Metadata Stor
e in the idle connection pool.
  kube_rbac_proxy_limit_memory    512Mi
string   Memory limit for kube-rbac-proxy container in the metadata-store-app d
eployment
  kubernetes_distribution
string   Kubernetes platform distribution where the metadata-store is being ins
talled on. Accepted values: ["", "openshift"]
  log_level                       default
string   Sets the log level. This can be set to "minimum", "less", "default",
"more", "debug" or "trace". "minimum" currently does not output logs. "less" ou
tputs log configuration options only. "default" and "more" outputs API endpoint
access information. "debug" and "trade" outputs extended API endpoint

access information(such as body payload) and other debug information.
  tls.server.rfcCiphers           [TLS_AES_128_GCM_SHA256 TLS_AES_256_GCM_SHA38
4 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA3
84 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384]
array    List of cipher suites for the server. Values are from tls package cons
tants (https://golang.org/pkg/crypto/tls/#pkg-constants). If omitted, the defau
lt Go cipher suites will be used
  ingress_issuer                  tap-ingress-selfsigned
string   tap-ingress-selfsigned is the default value when installed via any TAP
profile. When installing only the metadata-store package, a ClusterIssuer needs
to be installed and its name needs to be specified as this value.
```

4. (Optional) Edit one of the deployment configurations by creating a configuration YAML with the custom configuration values you want. For example, if your environment does not support `LoadBalancer`, and you want to use `ClusterIP`, then create a `metadata-store-values.yaml` and configure the `app_service_type` property.

   ```
   ---
   app_service_type: "ClusterIP"
   ```

   See Deployment details and configuration for more information about configuration options.

   For information about ingress and custom domain name support, see Ingress support.

5. Install the package using `tanzu package install`.

```
tanzu package install metadata-store \
  --package metadata-store.apps.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file metadata-store-values.yaml
```

Where:

- `--values-file` is an optional flag. Only use it to customize the deployment configuration.
- `VERSION` is the package version number.

For example:

```
$ tanzu package install metadata-store \
  --package metadata-store.apps.tanzu.vmware.com \
  --version 1.0.2 \
  --namespace tap-install \
  --values-file metadata-store-values.yaml

- Installing package 'metadata-store.apps.tanzu.vmware.com'
/ Getting namespace 'tap-install'
- Getting package metadata for 'metadata-store.apps.tanzu.vmware.com'
/ Creating service account 'metadata-store-tap-install-sa'
/ Creating cluster admin role 'metadata-store-tap-install-cluster-role'
/ Creating cluster role binding 'metadata-store-tap-install-cluster-rolebindin
g'
/ Creating secret 'metadata-store-tap-install-values'
| Creating package resource
- Package install status: Reconciling

Added installed package 'metadata-store' in namespace 'tap-install'
```

Overview of Tanzu Application Platform GUI

Tanzu Application Platform GUI (commonly called TAP GUI) is a tool for your developers to view your applications and services running for your organization. This portal provides a central location in which you can view dependencies, relationships, technical documentation, and the service status.

Tanzu Application Platform GUI is built from the Cloud Native Computing Foundation's project Backstage.

Tanzu Application Platform GUI consists of the following components:

- **Your organization catalog:**

  The catalog serves as the primary visual representation of your running services (components) and applications (systems).

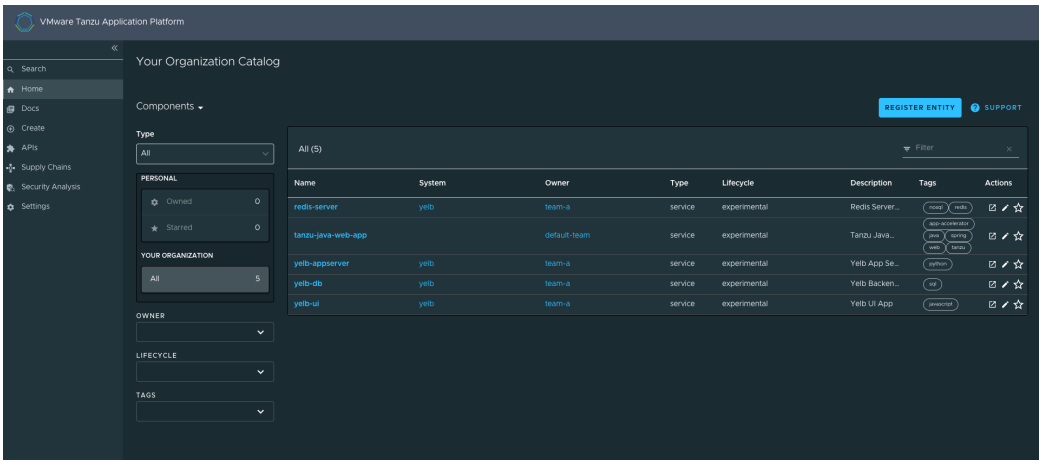
- **Tanzu Application Platform GUI plug-ins:**

  These plug-ins expose capabilities regarding specific Tanzu Application Platform tools. Initially the included plug-ins are:

  - Runtime Resources Visibility
  - Application Live View
  - Application Accelerator
  - API Documentation
  - Supply Chain Choreographer

- **TechDocs:**

  This plug-in enables you to store your technical documentation in Markdown format in a source-code repository and display it alongside the relevant catalog entries.



- **Search:**

  This plug-in enables you to search your organization's catalog, including domains, systems, components, APIs, accelerators, and TechDocs.

- **A Git repository:**

  Tanzu Application Platform GUI stores the following in a Git repository:

  - The structure for your application catalog.
  - Your technical documentation about the catalog items, if you enable Tanzu Application Platform GUI TechDocs capabilities.

You can host the structure for your application catalog and your technical documentation in the same repository as your source code.

Overview of Tanzu Application Platform GUI

Tanzu Application Platform GUI (commonly called TAP GUI) is a tool for your developers to view your applications and services running for your organization. This portal provides a central location in which you can view dependencies, relationships, technical documentation, and the service status.

Tanzu Application Platform GUI is built from the Cloud Native Computing Foundation's project Backstage.

Tanzu Application Platform GUI consists of the following components:

- **Your organization catalog:**

  The catalog serves as the primary visual representation of your running services (components) and applications (systems).

- **Tanzu Application Platform GUI plug-ins:**

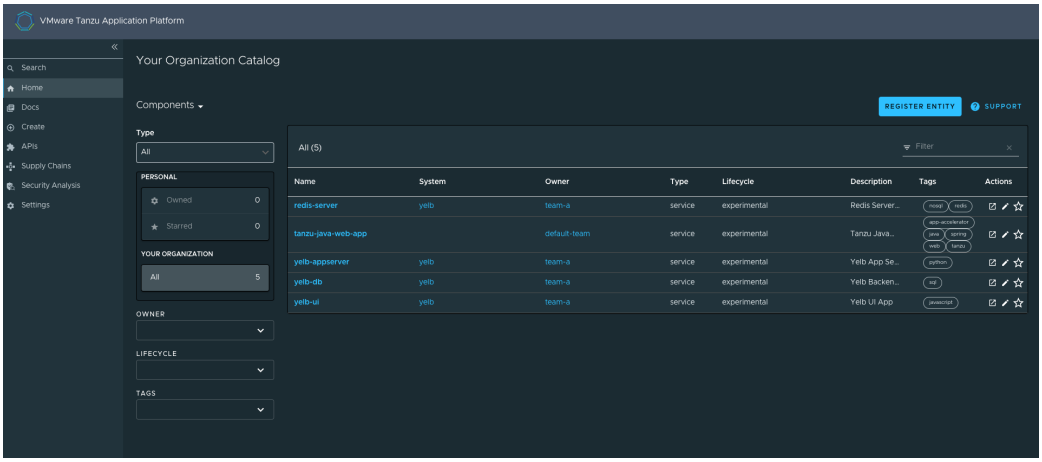
  These plug-ins expose capabilities regarding specific Tanzu Application Platform tools. Initially the included plug-ins are:

  - Runtime Resources Visibility
  - Application Live View
  - Application Accelerator
  - API Documentation

    - Supply Chain Choreographer

- **TechDocs:**

  This plug-in enables you to store your technical documentation in Markdown format in a source-code repository and display it alongside the relevant catalog entries.



- **Search:**

  This plug-in enables you to search your organization's catalog, including domains, systems, components, APIs, accelerators, and TechDocs.

- **A Git repository:**

  Tanzu Application Platform GUI stores the following in a Git repository:

    - The structure for your application catalog.
    - Your technical documentation about the catalog items, if you enable Tanzu Application Platform GUI TechDocs capabilities.

You can host the structure for your application catalog and your technical documentation in the same repository as your source code.

Install Tanzu Application Platform GUI

This topic tells you how to install Tanzu Application Platform GUI (commonly called TAP GUI) from the Tanzu Application Platform package repository.

> **Note**
>
> Follow the steps in this topic if you do not want to use a profile to install Tanzu Application Platform GUI. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Tanzu Application Platform GUI:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see the Tanzu Application Platform Prerequisites.
- Create a Git repository for Tanzu Application Platform GUI software catalogs, with a token allowing read access. Supported Git infrastructure includes:
    - GitHub

    - GitLab
    - Azure DevOps
- Install Tanzu Application Platform GUI Blank Catalog
    1. Go to the Tanzu Application Platform section of VMware Tanzu Network.
    2. Under the list of available files to download, open the **tap-gui-catalogs-latest** folder.
    3. Extract Tanzu Application Platform GUI Blank Catalog to your Git repository. This serves as the configuration location for your organization's Catalog inside Tanzu Application Platform GUI.

Procedure

To install Tanzu Application Platform GUI on a compliant Kubernetes cluster:

1. List version information for the package by running:

    ```
    tanzu package available list tap-gui.tanzu.vmware.com --namespace tap-install
    ```

    For example:

    ```
    $ tanzu package available list tap-gui.tanzu.vmware.com --namespace tap-install
    - Retrieving package versions for tap-gui.tanzu.vmware.com...
      NAME                      VERSION     RELEASED-AT
      tap-gui.tanzu.vmware.com  1.0.1       2022-01-10T13:14:23Z
    ```

2. (Optional) Make changes to the default installation settings by running:

    ```
    tanzu package available get tap-gui.tanzu.vmware.com/VERSION-NUMBER --values-sc
    hema --namespace \
    tap-install
    ```

    Where `VERSION-NUMBER` is the number you discovered previously. For example, `1.0.1`.

    For more information about values schema options, see the individual product documentation.

3. Create `tap-gui-values.yaml` and paste in the following YAML:

    ```
    ingressEnabled: true
    ingressDomain: "INGRESS-DOMAIN"
    app_config:
      catalog:
        locations:
          - type: url
            target: https://GIT-CATALOG-URL/catalog-info.yaml
    ```

    Where:

    - `INGRESS-DOMAIN` is the subdomain for the host name that you point at the `tanzu-shared-ingress` service's External IP address.
    - `GIT-CATALOG-URL` is the path to the `catalog-info.yaml` catalog definition file. It is from either the included Blank catalog (provided as an additional download named **Blank Tanzu Application Platform GUI Catalog**) or a Backstage-compliant catalog that you've already built and posted on the Git infrastructure specified in Adding Tanzu Application Platform GUI integrations.

4. Install the package by running:

```
tanzu package install tap-gui \
 --package tap-gui.tanzu.vmware.com \
 --version VERSION -n tap-install \
 --values-file tap-gui-values.yaml
```

Where `VERSION` is the version that you want. For example, `1.0.1`.

For example:

```
$ tanzu package install tap-gui --package tap-gui.tanzu.vmware.com --version 1.
0.1 -n \
tap-install --values-file tap-gui-values.yaml
- Installing package 'tap-gui.tanzu.vmware.com'
| Getting package metadata for 'tap-gui.tanzu.vmware.com'
| Creating service account 'tap-gui-default-sa'
| Creating cluster admin role 'tap-gui-default-cluster-role'
| Creating cluster role binding 'tap-gui-default-cluster-rolebinding'
| Creating secret 'tap-gui-default-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'tap-gui' in namespace 'tap-install'
```

5. Verify that the package installed by running:

```
tanzu package installed get tap-gui -n tap-install
```

For example:

```
$ tanzu package installed get tap-gui -n tap-install
| Retrieving installation details for cc...
NAME:                    tap-gui
PACKAGE-NAME:            tap-gui.tanzu.vmware.com
PACKAGE-VERSION:         1.0.1
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

6. To access Tanzu Application Platform GUI, use the service you exposed in the `service_type` field in the values file.

Runtime configuration options for Tanzu Application Platform GUI

You can provide a series of options to the Tanzu Application Platform GUI (commonly called TAP GUI) package to configure it and do some basic runtime customization.

Identify the Tanzu Application Platform GUI version you have available

From the Tanzu CLI, discover the Tanzu Application Platform GUI package version that is available to configure by running:

```
tanzu package available get tap-gui.tanzu.vmware.com -n INSTALL-NAMESPACE
```

Where `INSTALL-NAMESPACE` is the namespace in which you configured the Tanzu Application Platform installation. In most cases the namespace is `tap-install`.

For example:

```
$ tanzu package available get tap-gui.tanzu.vmware.com -n tap-install

NAME:                    tap-gui.tanzu.vmware.com
DISPLAY-NAME:            Tanzu Application Platform GUI
CATEGORIES:
SHORT-DESCRIPTION:       web app graphical user interface for Tanzu Application Platfor
m
LONG-DESCRIPTION:        web app graphical user interface for Tanzu Application Platfor
m
PROVIDER:                VMware
MAINTAINERS:             - name: VMware
SUPPORT-DESCRIPTION:     https://tanzu.vmware.com/support

  VERSION  RELEASED-AT
  1.7.6    2023-10-17 00:25:21 +0000 UTC
```

Display the possible values options for Tanzu Application Platform GUI

From the Tanzu CLI, identify possible values options for Tanzu Application Platform GUI by running:

```
tanzu package available get tap-gui.tanzu.vmware.com/VERSION --values-schema -n INSTAL
L-NAMESPACE
```

Where:

- `VERSION` is the Tanzu Application Platform GUI package version you learned earlier
- `INSTALL-NAMESPACE` is the namespace in which you configured the Tanzu Application Platform installation. In most cases the namespace is `tap-install`.

For example:

```
$ tanzu package available get tap-gui.tanzu.vmware.com/1.7.6 --values-schema -n tap-in
stall

  KEY                                                              DEFAULT   TYPE
DESCRIPTION
  #Details of all the possible configuration values
  ...
```

Customize the Tanzu Application Platform GUI portal

This section describes how to customize the Tanzu Application Platform GUI portal.

Customize branding

To customize the branding in your portal, you can choose the name of the portal and the logo for it. To make these customizations:

1. Provide additional configuration parameters to the `app_config` section of `tap-values.yaml`:

   ```
   tap_gui:
     app_config:
       customize:
         custom_logo: 'BASE-64-IMAGE'
         custom_name: 'PORTAL-NAME'
   ```

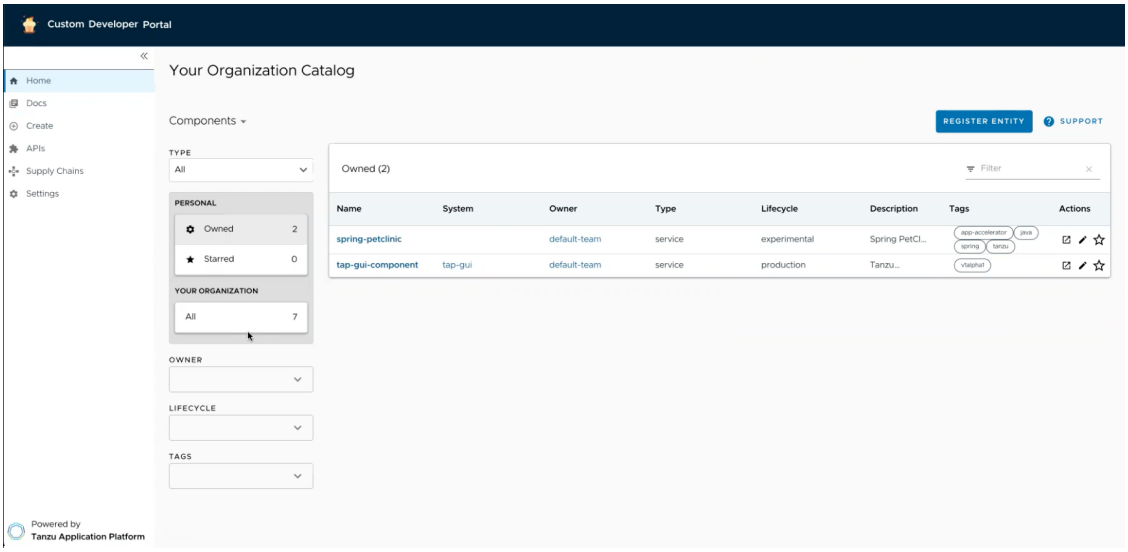
Where:

- `BASE-64-IMAGE` is the image encoded in base64. A 512-pixel by 512-pixel PNG image with a transparent background is optimal.
- `PORTAL-NAME` is the name of your portal, such as `Our Custom Developer Experience Portal`.

2. Reinstall your Tanzu Application Platform GUI package by following steps in Upgrading Tanzu Application Platform.

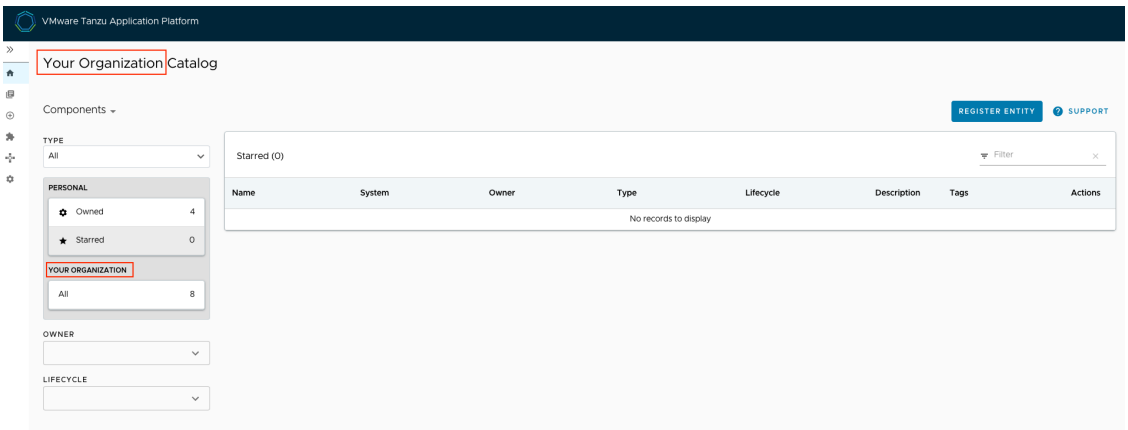
After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in any of the supplied images encoded in base64 or in your choice of portal name, Tanzu Application Platform GUI reverts to the original branding template.



Customize the Software Catalog page

You can customize the name of your organization on the Software Catalog page of Tanzu Application Platform GUI portal. By default, the portal displays **Your Organization** next to **Catalog** and in the selection box.



Customize the name of the organization

To customize the name of the organization for the software catalog in your portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

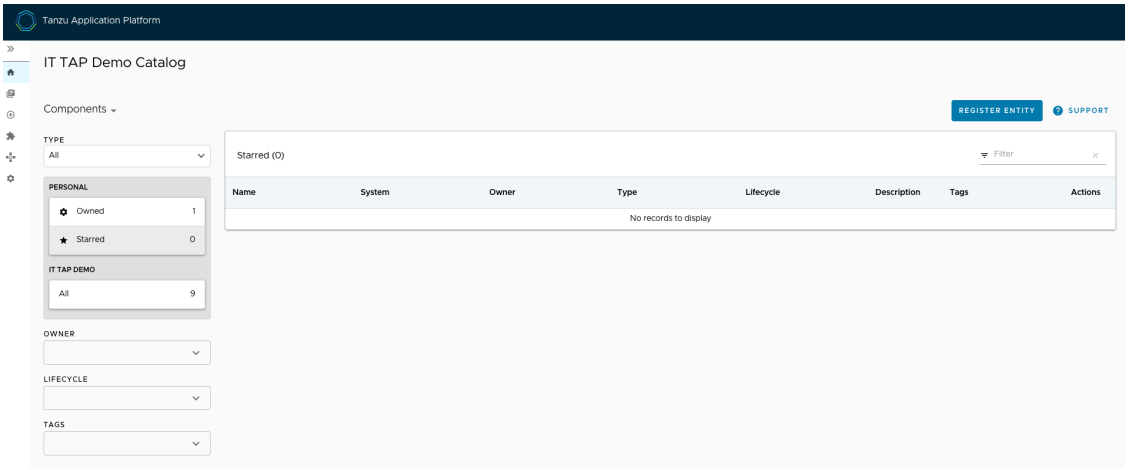
```
tap_gui:
  app_config:
    organization:
      name: 'ORG-NAME'
```

Where `ORG-NAME` is the name of your organization for the software catalog, such as `Our Organization Name`. You don't need to add `Catalog` to the `ORG-NAME`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in the provided configuration parameters, Tanzu Application Platform GUI reverts to the original organization name.



Prevent changes to the software catalog

You can deactivate the **Register Entity** button to prevent a user from making changes to the software catalog, including registering and deregistering locations. To do so, add `readonly: true` to the `catalog` section in `tap-values.yaml`, as in this example:

```
tap_gui:
  app_config:
    catalog:
      readonly: true
```

Customize the Authentication page

To customize the portal name on the **Authentication** page and the name of the browser tab for Tanzu Application Platform GUI:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

   ```
   tap_gui:
     app_config:
       app:
         title: 'CUSTOM-TAB-NAME'
   ```

   Where `CUSTOM-TAB-NAME` is the name on the Authentication page and the browser tab of your portal, such as `Our Organization Full Name`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

Customize the default view

You can set your default route when the user is accessing your portal. Without this customization, when the user accesses the Tanzu Application Platform GUI URL, it displays the list of owned components of the software catalog.

To change the default view:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

   ```
   tap_gui:
     app_config:
       customize:
         default_route: 'YOUR-PREFERRED-ROUTE'
   ```

   Where `YOUR-PREFERRED-ROUTE` is the path to the route that the portal uses by default. For example, you can type `/catalog?filters%5Bkind%5D=component&filters%5Buser%5D=all` to show all components of the software catalog instead of defaulting to owned components. As another example, you can type `/create` to show Application Accelerator when the portal starts.

   Caution

   Tanzu Application Platform GUI redirects you to `tap-gui.INGRESS-DOMAIN/YOUR-PREFERRED-ROUTE` even if there is an error in `YOUR-PREFERRED-ROUTE`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

Customize security banners

You can instruct Tanzu Application Platform GUI to create security banners on the top and bottom of the page. To add security banners to Tanzu Application Platform GUI:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file, as in the following example:

   ```
   tap_gui:
     app_config:
       customize:
         banners:
           text: 'CUSTOM-TEXT'
           color: 'OPTIONAL-CUSTOM-TEXT-COLOR'
           bg: 'CUSTOM-BACKGROUND-COLOR'
           link: 'OPTIONAL-LINK'
   ```

   Where:

- `CUSTOM-TEXT` is the text that is displayed in the banner. Keep this text short to accommodate various screen sizes.
- `OPTIONAL-CUSTOM-TEXT-COLOR` is the color of the text displayed in the banner. Setting this is optional. It accepts CSS colors, such as `#ffffff`. The default color is `#FFFFFF`.
- `CUSTOM-BACKGROUND-COLOR` is the color of the banner itself. Setting this is optional. It accepts CSS colors, such as `#ffffff`. The default color is `#C23B2E`
- `OPTIONAL-LINK` is the link to which your text redirects. Setting this is optional.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, the customized version of your portal is displayed.

Customize the Tanzu Application Platform GUI portal

This section describes how to customize the Tanzu Application Platform GUI portal.

Customize branding

To customize the branding in your portal, you can choose the name of the portal and the logo for it. To make these customizations:

1. Provide additional configuration parameters to the `app_config` section of `tap-values.yaml`:

   ```
   tap_gui:
     app_config:
       customize:
         custom_logo: 'BASE-64-IMAGE'
         custom_name: 'PORTAL-NAME'
   ```

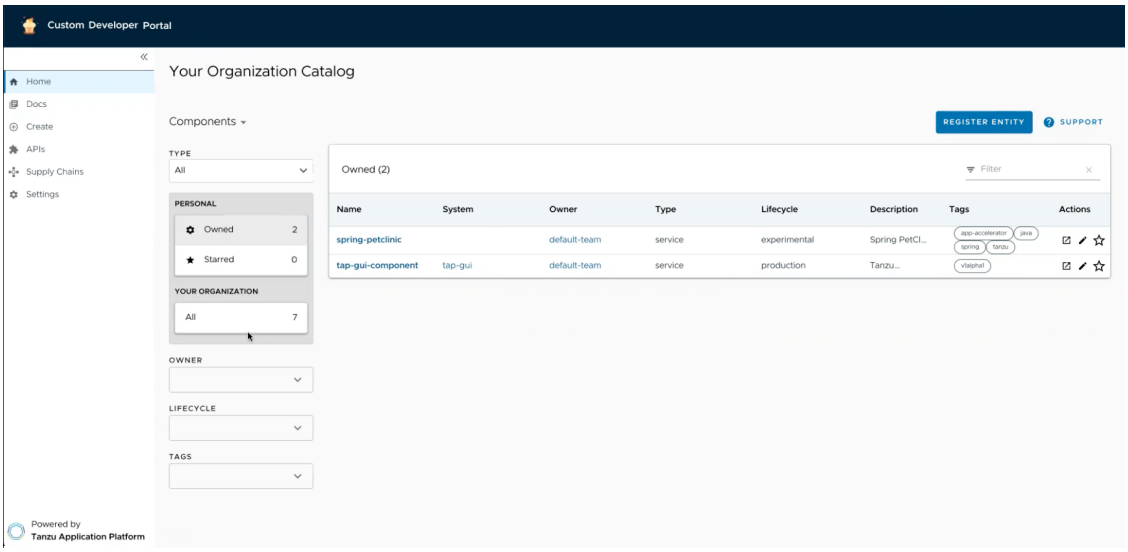
   Where:

   - `BASE-64-IMAGE` is the image encoded in base64. A 512-pixel by 512-pixel PNG image with a transparent background is optimal.
   - `PORTAL-NAME` is the name of your portal, such as `Our Custom Developer Experience Portal`.

2. Reinstall your Tanzu Application Platform GUI package by following steps in Upgrading Tanzu Application Platform.

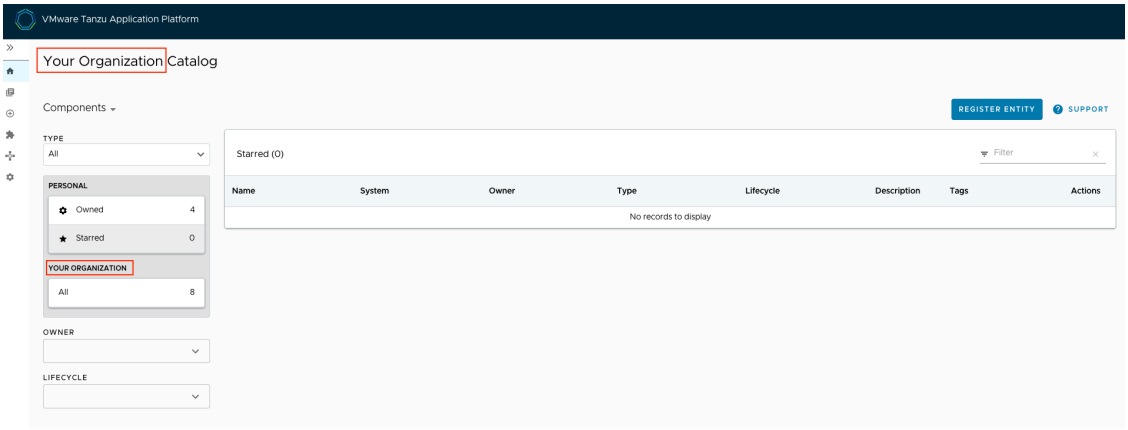
After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in any of the supplied images encoded in base64 or in your choice of portal name, Tanzu Application Platform GUI reverts to the original branding template.



Customize the Software Catalog page

You can customize the name of your organization on the Software Catalog page of Tanzu Application Platform GUI portal. By default, the portal displays **Your Organization** next to **Catalog** and in the selection box.



Customize the name of the organization

To customize the name of the organization for the software catalog in your portal:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

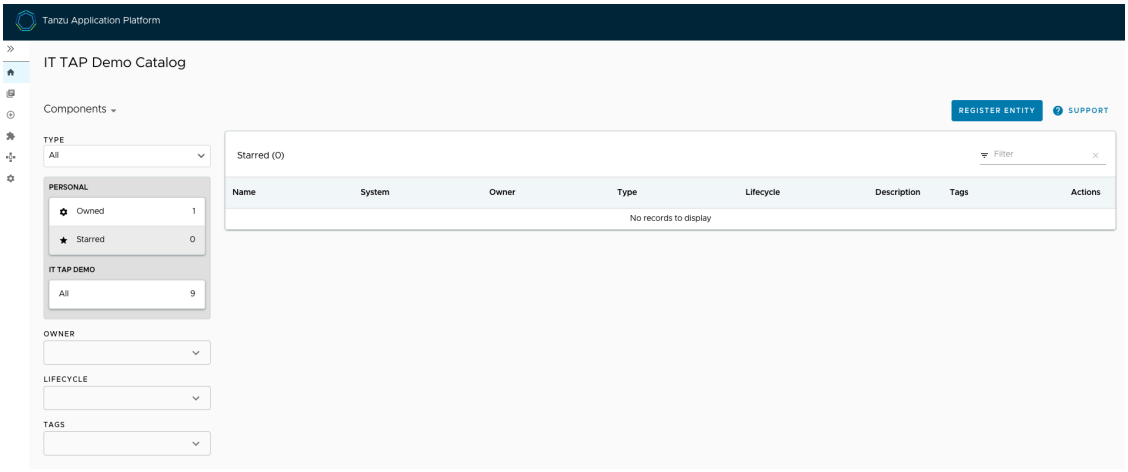
   ```
   tap_gui:
     app_config:
       organization:
         name: 'ORG-NAME'
   ```

   Where `ORG-NAME` is the name of your organization for the software catalog, such as `Our Organization Name`. You don't need to add `Catalog` to the `ORG-NAME`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

If there is an error in the provided configuration parameters, Tanzu Application Platform GUI reverts to the original organization name.



Prevent changes to the software catalog

You can deactivate the **Register Entity** button to prevent a user from making changes to the software catalog, including registering and deregistering locations. To do so, add `readonly: true` to the `catalog` section in `tap-values.yaml`, as in this example:

```
tap_gui:
  app_config:
    catalog:
      readonly: true
```

Customize the Authentication page

To customize the portal name on the **Authentication** page and the name of the browser tab for Tanzu Application Platform GUI:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

   ```
   tap_gui:
     app_config:
       app:
         title: 'CUSTOM-TAB-NAME'
   ```

   Where `CUSTOM-TAB-NAME` is the name on the Authentication page and the browser tab of your portal, such as `Our Organization Full Name`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

Customize the default view

You can set your default route when the user is accessing your portal. Without this customization, when the user accesses the Tanzu Application Platform GUI URL, it displays the list of owned components of the software catalog.

To change the default view:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file:

   ```
   tap_gui:
     app_config:
       customize:
         default_route: 'YOUR-PREFERRED-ROUTE'
   ```

   Where `YOUR-PREFERRED-ROUTE` is the path to the route that the portal uses by default. For example, you can type `/catalog?filters%5Bkind%5D=component&filters%5Buser%5D=all` to show all components of the software catalog instead of defaulting to owned components. As another example, you can type `/create` to show Application Accelerator when the portal starts.

   Caution

   Tanzu Application Platform GUI redirects you to `tap-gui.INGRESS-DOMAIN/YOUR-PREFERRED-ROUTE` even if there is an error in `YOUR-PREFERRED-ROUTE`.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

After the updated values configuration file is applied in Tanzu Application Platform GUI, you see the customized version of your portal.

Customize security banners

You can instruct Tanzu Application Platform GUI to create security banners on the top and bottom of the page. To add security banners to Tanzu Application Platform GUI:

1. Provide additional configuration parameters to the `app_config` section of your `tap-values.yaml` file, as in the following example:

   ```
   tap_gui:
     app_config:
       customize:
         banners:
           text: 'CUSTOM-TEXT'
           color: 'OPTIONAL-CUSTOM-TEXT-COLOR'
           bg: 'CUSTOM-BACKGROUND-COLOR'
           link: 'OPTIONAL-LINK'
   ```

   Where:

   - `CUSTOM-TEXT` is the text that is displayed in the banner. Keep this text short to accommodate various screen sizes.
   - `OPTIONAL-CUSTOM-TEXT-COLOR` is the color of the text displayed in the banner. Setting this is optional. It accepts CSS colors, such as `#ffffff`. The default color is `#FFFFFF`.
   - `CUSTOM-BACKGROUND-COLOR` is the color of the banner itself. Setting this is optional. It accepts CSS colors, such as `#ffffff`. The default color is `#C23B2E`
   - `OPTIONAL-LINK` is the link to which your text redirects. Setting this is optional.

2. Reinstall your Tanzu Application Platform GUI package by following the steps in Upgrading Tanzu Application Platform.

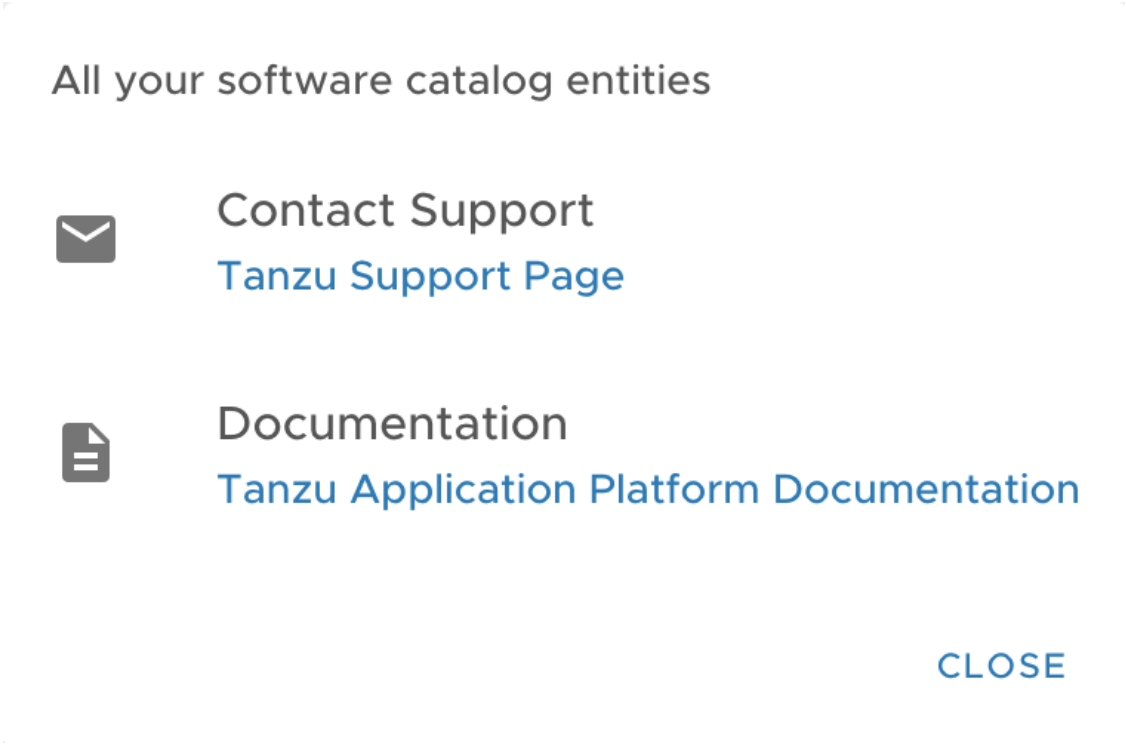
After the updated values configuration file is applied in Tanzu Application Platform GUI, the customized version of your portal is displayed.

Customize the Support menu

This topic describes how to customize the support menu.

Overview

Many important pages of Tanzu Application Platform GUI have a **Support** button that displays a pop-out menu. This menu contains a one-line description of the page the user is looking at, and a list of support item groupings.



As standard, there are two support item groupings:

- Contact Support, which is marked with an **email** icon and contains a link to VMware Tanzu's support portal.
- Documentation, which is marked with a **docs** icon and contains a link to the Tanzu Application Platform documentation that you are currently reading.

Customizing

The set of support item groupings is completely customizable. However, you might want to offer custom in-house links for your Tanzu Application Platform users rather than simply sending them to VMware support and documentation. You can provide this configuration by using your `tap-values.yaml`. Here is a configuration snippet, which produces the default support menu:

```
tap_gui:
  app_config:
    app:
      support:
        url: https://tanzu.vmware.com/support
        items:
          - title: Contact Support
```

```
            icon: email
            links:
              - url: https://tanzu.vmware.com/support
                title: Tanzu Support Page
          - title: Documentation
            icon: docs
            links:
              - url: https://docs.vmware.com/en/VMware-Tanzu-Application-Platform/inde
x.html
                title: Tanzu Application Platform Documentation
```

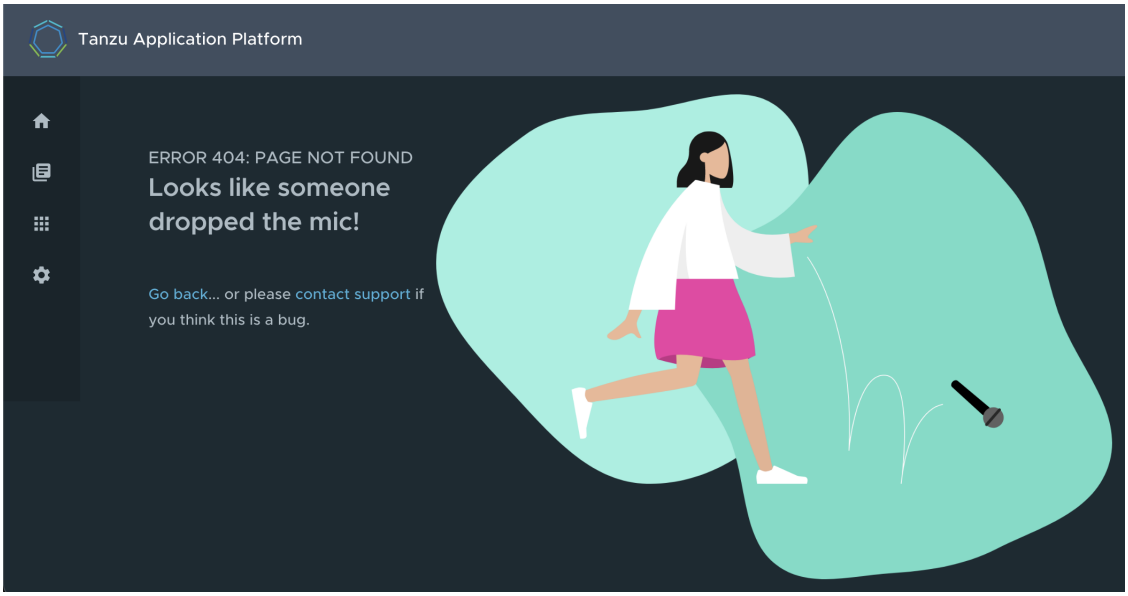
Structure of the support configuration

URL

The `url` field under the `support` section, for example,

```
      support:
        url: https://tanzu.vmware.com/support
```

provides the address of the **contact support** link that appears on error pages.



Items

The `items` field under the `support` section, for example, provides the set of support item groupings to display when the support menu is expanded.

Title

The `title` field on a support item grouping, for example,

```
        items:
          - title: Contact Support
```

provides the label for the grouping.

Icon

The `icon` field on a support item grouping, for example,

```
      items:
        - icon: email
```

provides the icon to use for that grouping. The valid choices are:

- `brokenImage`
- `catalog`
- `chat`
- `dashboard`
- `docs`
- `email`
- `github`
- `group`
- `help`
- `user`
- `warning`

Links

The `links` field on a support item grouping, for example,

```
      items:
        - links:
            - url: https://tanzu.vmware.com/support
              title: Tanzu Support Page
```

is a list of YAML objects that render as links. Each link has the text given by the `title` field and links to the value of the `url` field.

Access Tanzu Application Platform GUI

This topic tells you how to access Tanzu Application Platform GUI (commonly called TAP GUI) by using one of the following methods:

- Access with the LoadBalancer method (default)
- Access with the shared Ingress method

Access with the LoadBalancer method (default)

1. Verify that you specified the `service_type` for Tanzu Application Platform GUI in `tap-values.yaml`, as in this example:

   ```
   tap_gui:
     service_type: LoadBalancer
   ```

2. Obtain the external IP address of your LoadBalancer by running:

   ```
   kubectl get svc -n tap-gui
   ```

3. Access Tanzu Application Platform GUI by using the external IP address with the default port of 7000. It has the following form:

```
http://EXTERNAL-IP:7000
```

Where `EXTERNAL-IP` is the external IP address of your LoadBalancer.

Access with the shared Ingress method

The Ingress method of access for Tanzu Application Platform GUI uses the shared `tanzu-system-ingress` instance of Contour that is installed as part of the Profile installation.

1. The Ingress method of access requires that you have a DNS host name that you can point at the External IP address of the `envoy` service that the shared `tanzu-system-ingress` uses. Retrieve this IP address by running:

   ```
   kubectl get service envoy -n tanzu-system-ingress
   ```

   This returns a value similar to this example:

   ```
   $ kubectl get service envoy -n tanzu-system-ingress
   NAME    TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
   AGE
   envoy   LoadBalancer   10.0.242.171   40.118.168.232   80:31389/TCP,443:31780/T
   CP   27h
   ```

   The IP address in the `EXTERNAL-IP` field is the one that you point a DNS host record to. Tanzu Application Platform GUI prepends `tap-gui` to your provided subdomain. This makes the final host name `tap-gui.YOUR-SUBDOMAIN`. You use this host name in the appropriate fields in the `tap-values.yaml` file mentioned later.

2. Specify parameters in `tap-values.yaml` related to Ingress. For example:

   ```
   shared:
     ingress_domain: "example.com"
   ```

3. Update your other host names in the `tap_gui` section of your `tap-values.yaml` with the new host name. For example:

   ```
   shared:
     ingress_domain: "example.com"

   tap_gui:
   # Existing tap-values.yaml above
     app_config:
       app:
         baseUrl: http://tap-gui.example.com # No port needed with Ingress
       integrations:
         github: # Other are integrations available
           - host: github.com
             token: GITHUB-TOKEN
       catalog:
         locations:
           - type: url
             target: https://GIT-CATALOG-URL/catalog-info.yaml
       backend:
         baseUrl: http://tap-gui.example.com # No port needed with Ingress
         cors:
           origin: http://tap-gui.example.com # No port needed with Ingress
   ```

4. Update your package installation with your changed `tap-values.yaml` file by running:

   ```
   tanzu package installed update tap --package tap.tanzu.vmware.com --version VER
   SION-NUMBER \
   ```

```
--values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.

5. Use a web browser to access Tanzu Application Platform GUI at the host name that you provided.

Catalog operations

The software catalog setup procedures in this topic make use of Backstage. For more information about Backstage, see the Backstage documentation.

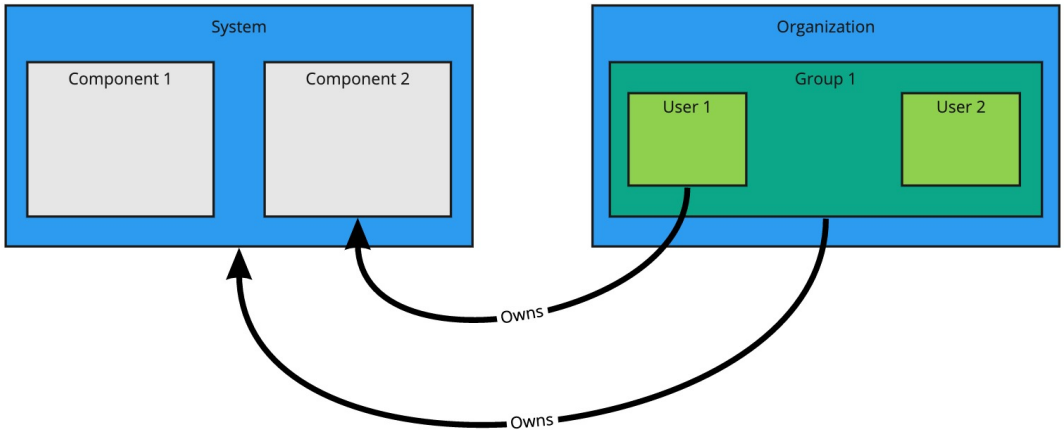
Adding catalog entities

This section describes how you can format your own catalog. Creating catalogs consists of building metadata YAML files stored together with the code. This information is read from a Git-compatible repository consisting of these YAML catalog definition files. Changes made to the catalog definitions on your Git infrastructure are automatically reflected every 200 seconds or when manually registered.

For each catalog entity kind you create, there is a file format you must follow. For information about all types of entities, see the Backstage documentation.

You can use the example blank catalog described in the Tanzu Application Platform GUI prerequisites as a foundation for creating user, group, system, and main component YAML files.

The organization contains Group 1, and Group 1 contains Users 1 and 2. System contains Components 1 and 2. User 1 owns Component 2. Group 1 owns System.



Users and groups

A user entity describes a specific person and is used for identity purposes. Users are members of one or more groups. A group entity describes an organizational team or unit.

Users and groups have different descriptor requirements in their descriptor files:

- User descriptor files require `apiVersion`, `kind`, `metadata.name`, and `spec.memberOf`.
- Group descriptor files require `apiVersion`, `kind`, and `metadata.name`. They also require `spec.type` and `spec.children` where `spec.children` is another group.

To link a logged-in user to a user entity, include the optional `spec.profile.email` field.

Sample user entity:

```
apiVersion: backstage.io/v1alpha1
kind: User
```

```
metadata:
  name: default-user
spec:
  profile:
    displayName: Default User
    email: guest@example.com
    picture: https://avatars.dicebear.com/api/avataaars/guest@example.com.svg?backgrou
nd=%23fff
  memberOf: [default-team]
```

Sample group entity:

```
apiVersion: backstage.io/v1alpha1
kind: Group
metadata:
  name: default-team
  description: Default Team
spec:
  type: team
  profile:
    displayName: Default Team
    email: team-a@example.com
    picture: https://avatars.dicebear.com/api/identicon/team-a@example.com.svg?backgro
und=%23fff
  parent: default-org
  children: []
```

For more information about user entities and group entities, see the Backstage documentation.

Systems

A system entity is a collection of resources and components.

System descriptor files require values for `apiVersion`, `kind`, `metadata.name`, and also `spec.owner` where `spec.owner` is a user or group.

A system has components when components specify the system name in the field `spec.system`.

Sample system entity:

```
apiVersion: backstage.io/v1alpha1
kind: System
metadata:
  name: backstage
  description: Tanzu Application Platform GUI System
spec:
  owner: default-team
```

For more information about system entities, see the Backstage documentation.

Components

A component describes a software component, or what might be described as a unit of software.

Component descriptor files require values for `apiVersion`, `kind`, `metadata.name`, `spec.type`, `spec.lifecycle`, and `spec.owner`.

Some useful optional fields are `spec.system` and `spec.subcomponentOf`, both of which link a component to an entity that it is part of.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: backstage-component
```

```
  description: Tanzu Application Platform GUI Component
  annotations:
    'backstage.io/kubernetes-label-selector': 'app=backstage' #Identifies the Kubernet
es objects that make up this component
    'backstage.io/techdocs-ref': dir:. #TechDocs label
spec:
  type: service
  lifecycle: alpha
  owner: default-team
  system: backstage
```

For more information about component entities, see the Backstage documentation.

Update software catalogs

The following procedures describe how to update software catalogs.

Register components

To update your software catalog with new entities without re-deploying the entire `tap-gui` package:

1. Go to your **Software Catalog** page.
2. Click **Register Entity** at the top-right of the page.
3. Enter the full path to link to an existing entity file and start tracking your entity.
4. Import the entities and view them in your **Software Catalog** page.

Deregister components

To deregister an entity:

1. Go to your **Software Catalog** page.
2. Select the entity to deregister, such as component, group, or user.
3. Click the three dots at the top-right of the page and then click **Unregister...**.

Add or change organization catalog locations

To add or change organization catalog locations, you can use static configuration or you can use `GitLabDiscoveryProcessor` to discover and register catalog entities that match the configured path.

Use static configuration
To use static configuration to add or change catalog locations:

1. Update components by changing the catalog location in either the `app_config` section of `tap-gui-values.yaml` or the custom values file you used when installing. For example:

   ```
   tap_gui:
     app_config:
       catalog:
         locations:
           - type: url
             target: UPDATED-CATALOG-LOCATION
   ```

2. Register components by adding the new catalog location in either the `app_config` section of `tap-gui-values.yaml` or the custom values file you used when installing. For example:

```
tap_gui:
  app_config:
    catalog:
      locations:
        - type: url
          target: EXISTING-CATALOG-LOCATION
        - type: url
          target: EXTRA-CATALOG-LOCATION
```

When targeting GitHub, don't write the raw URL. Instead, use the URL that appears when you navigate to the file in the browser. The catalog processor cannot set up the files properly if you use the raw URL.

- Example raw URL:
  `https://raw.githubusercontent.com/user/repo/catalog.yaml`
- Example target URL: `https://github.com/user/repo/blob/main/catalog.yaml`

When targeting GitLab, use a scoped route to the catalog file. This is a route with the `/-/` separator after the project name. If you don't use a scoped route, your entity fails to appear in the catalog.

- Example unscoped URL:
  `https://gitlab.com/group/project/blob/main/catalog.yaml`
- Example target URL:
  `https://gitlab.com/group/project/-/blob/main/catalog.yaml`

For more information about static catalog configuration, see the Backstage documentation.

Use GitLabDiscoveryProcessor

To use `GitLabDiscoveryProcessor` to discover and register catalog entities:

1. Use `type: gitlab-discovery` to make `GitLabDiscoveryProcessor` crawl the GitLab instance to discover and register catalog entities that match the configured path. For more information, see the Backstage documentation.
2. Update the package to include the catalog:
   - If you installed Tanzu Application Platform GUI by using a profile, run:

     ```
     tanzu package installed update tap \
     --package tap.tanzu.vmware.com \
     --version PACKAGE-VERSION \
     --values-file tap-values.yaml \
     --namespace tap-install
     ```

   - If you installed Tanzu Application Platform GUI as an individual package, run:

     ```
     tanzu package installed update tap-gui \
     --package tap-gui.tanzu.vmware.com \
     --version PACKAGE-VERSION \
     --values-file tap-gui-values.yaml \
     --namespace tap-install
     ```

3. Verify the status of this update by running:

   ```
   tanzu package installed list -n tap-install
   ```

Install demo apps and their catalogs

To set up one of the demos, you can choose a blank catalog or a sample catalog.

Yelb system

The Yelb demo catalog in GitHub includes all the components that make up the Yelb system and the default Backstage components.

Install Yelb

To install Yelb:

1. Download the necessary file for running the Yelb application itself from GitHub.
2. Install the application on the Kubernetes cluster that you used for Tanzu Application Platform. Preserve the metadata labels on the Yelb application objects.

Install the Yelb catalog

To install the Yelb catalog:

1. In Tanzu Network, select your release from the drop-down menu.
2. Click **tap-gui-catalogs-latest** > **Tanzu Application Platform GUI Yelb Catalog** and download the catalog.
3. Unpack the downloaded TAR archive to a local drive.
4. Follow the earlier steps for Register components to register the `catalog-info.yaml` in the root of the unpacked archive and register all the catalog entities that constitute the Yelb system.

View resources on multiple clusters in Tanzu Application Platform GUI

You can configure Tanzu Application Platform GUI (commonly called TAP GUI) to retrieve Kubernetes object details from multiple clusters and then surface those details in the various Tanzu Application Platform GUI plug-ins.

Important

In this topic the terms `Build`, `Run`, and `View` describe the cluster's roles and distinguish which steps to apply to which cluster.

`Build` clusters are where the code is built and packaged, ready to be run.

`Run` clusters are where the Tanzu Application Platform workloads themselves run.

`View` clusters are where the Tanzu Application Platform GUI is run from.

In multicluster configurations, these can be separate clusters. However, in many configurations these can also be the same cluster.

Set up a Service Account to view resources on a cluster

To view resources on the `Build` or `Run` clusters, create a service account on the `View` cluster that can `get`, `watch`, and `list` resources on those clusters.

You first create a `ClusterRole` with these rules and a `ServiceAccount` in its own `Namespace`, and then bind the `ClusterRole` to the `ServiceAccount`. Depending on your topology, not every cluster

has all of the following objects. For example, the `Build` cluster doesn't have any of the `serving.knative.dev` objects, by design, because it doesn't run the workloads themselves. You can edit the following object lists to reflect your topology.

To set up a Service Account to view resources on a cluster:

1. Copy this YAML content into a file called `tap-gui-viewer-service-account-rbac.yaml`.

```
apiVersion: v1
kind: Namespace
metadata:
  name: tap-gui
---
apiVersion: v1
kind: ServiceAccount
metadata:
  namespace: tap-gui
  name: tap-gui-viewer
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tap-gui-read-k8s
subjects:
- kind: ServiceAccount
  namespace: tap-gui
  name: tap-gui-viewer
roleRef:
  kind: ClusterRole
  name: k8s-reader
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: k8s-reader
rules:
- apiGroups: ['']
  resources: ['pods', 'pods/log', 'services', 'configmaps', 'limitranges']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['metrics.k8s.io']
  resources: ['pods']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['apps']
  resources: ['deployments', 'replicasets', 'statefulsets', 'daemonsets']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['autoscaling']
  resources: ['horizontalpodautoscalers']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['networking.k8s.io']
  resources: ['ingresses']
  verbs: ['get', 'watch', 'list']
- apiGroups: ['networking.internal.knative.dev']
  resources: ['serverlessservices']
  verbs: ['get', 'watch', 'list']
- apiGroups: [ 'autoscaling.internal.knative.dev' ]
  resources: [ 'podautoscalers' ]
  verbs: [ 'get', 'watch', 'list' ]
- apiGroups: ['serving.knative.dev']
  resources:
  - configurations
  - revisions
  - routes
  - services
  verbs: ['get', 'watch', 'list']
- apiGroups: ['carto.run']
```

```
  resources:
  - clusterconfigtemplates
  - clusterdeliveries
  - clusterdeploymenttemplates
  - clusterimagetemplates
  - clusterruntemplates
  - clustersourcetemplates
  - clustersupplychains
  - clustertemplates
  - deliverables
  - runnables
  - workloads
  verbs: ['get', 'watch', 'list']
- apiGroups: ['source.toolkit.fluxcd.io']
  resources:
  - gitrepositories
  verbs: ['get', 'watch', 'list']
- apiGroups: ['source.apps.tanzu.vmware.com']
  resources:
  - imagerepositories
  - mavenartifacts
  verbs: ['get', 'watch', 'list']
- apiGroups: ['conventions.apps.tanzu.vmware.com']
  resources:
  - podintents
  verbs: ['get', 'watch', 'list']
- apiGroups: ['kpack.io']
  resources:
  - images
  - builds
  verbs: ['get', 'watch', 'list']
- apiGroups: ['scanning.apps.tanzu.vmware.com']
  resources:
  - sourcescans
  - imagescans
  - scanpolicies
  - scantemplates
  verbs: ['get', 'watch', 'list']
- apiGroups: ['tekton.dev']
  resources:
  - taskruns
  - pipelineruns
  verbs: ['get', 'watch', 'list']
- apiGroups: ['kappctrl.k14s.io']
  resources:
  - apps
  verbs: ['get', 'watch', 'list']
- apiGroups: [ 'batch' ]
  resources: [ 'jobs', 'cronjobs' ]
  verbs: [ 'get', 'watch', 'list' ]
- apiGroups: ['conventions.carto.run']
  resources:
  - podintents
  verbs: ['get', 'watch', 'list']
- apiGroups: ['appliveview.apps.tanzu.vmware.com']
  resources:
  - resourceinspectiongrants
  verbs: ['get', 'watch', 'list', 'create']
```

This YAML content creates `Namespace`, `ServiceAccount`, `ClusterRole`, and `ClusterRoleBinding`.

2. On the `Build` and `Run` clusters, create `Namespace`, `ServiceAccount`, `ClusterRole`, and `ClusterRoleBinding` by running:

```
kubectl create -f tap-gui-viewer-service-account-rbac.yaml
```

3. Again, on the `Build` and `Run` clusters, discover the `CLUSTER_URL` and `CLUSTER_TOKEN` values.

v1.23 or earlier Kubernetes cluster

If you're watching a v1.23 or earlier Kubernetes cluster, run:

```
CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluste
r.server}')

CLUSTER_TOKEN=$(kubectl -n tap-gui get secret $(kubectl -n tap-gui get sa tap
-gui-viewer -o=json \
| jq -r '.secrets[0].name') -o=json \
| jq -r '.data["token"]' \
| base64 --decode)

echo CLUSTER_URL: $CLUSTER_URL
echo CLUSTER_TOKEN: $CLUSTER_TOKEN
```

v1.24 or later Kubernetes cluster

If you're watching a v1.24 or later Kubernetes cluster, run:

```
CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluste
r.server}')

kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: tap-gui-viewer
  namespace: tap-gui
  annotations:
    kubernetes.io/service-account.name: tap-gui-viewer
type: kubernetes.io/service-account-token
EOF

CLUSTER_TOKEN=$(kubectl -n tap-gui get secret tap-gui-viewer -o=json \
| jq -r '.data["token"]' \
| base64 --decode)

echo CLUSTER_URL: $CLUSTER_URL
echo CLUSTER_TOKEN: $CLUSTER_TOKEN
```

Note

You can create a short-lived token with the `kubectl create token` command if that is the preferred method. This method requires frequent token rotation.

4. (Optional) Configure the Kubernetes client to verify the TLS certificates presented by a cluster's API server. To do this, discover `CLUSTER_CA_CERTIFICATES` by running:

```
CLUSTER_CA_CERTIFICATES=$(kubectl config view --raw -o jsonpath='{.clusters[?
(@.name=="CLUSTER-NAME")].cluster.certificate-authority-data}')

echo CLUSTER_CA_CERTIFICATES: $CLUSTER_CA_CERTIFICATES
```

Where `CLUSTER-NAME` is your cluster name.

5. Record the `Build` and `Run` clusters' `CLUSTER_URL` and `CLUSTER_TOKEN` values for when you Update Tanzu Application Platform GUI to view resources on multiple clusters later.

Update Tanzu Application Platform GUI to view resources on multiple clusters

The clusters must be identified to Tanzu Application Platform GUI with the `ServiceAccount` token and the cluster Kubernetes control plane URL.

You must add a `kubernetes` section to the `app_config` section in the `tap-values.yaml` file that Tanzu Application Platform used when you installed it. This section must have an entry for each `Build` and `Run` cluster that has resources to view.

To do so:

1. Copy this YAML content into `tap-values.yaml`:

```
tap_gui:
## Previous configuration above
  app_config:
    kubernetes:
      serviceLocatorMethod:
        type: 'multiTenant'
      clusterLocatorMethods:
        - type: 'config'
          clusters:
          ## Cluster 1
            - url: CLUSTER-URL
              name: CLUSTER-NAME
              authProvider: serviceAccount
              serviceAccountToken: "CLUSTER-TOKEN"
              skipTLSVerify: true
              skipMetricsLookup: true
          ## Cluster 2+
            - url: CLUSTER-URL
              name: CLUSTER-NAME
              authProvider: serviceAccount
              serviceAccountToken: "CLUSTER-TOKEN"
              skipTLSVerify: true
              skipMetricsLookup: true
```

   Where:

   - `CLUSTER-URL` is the value you discovered earlier.
   - `CLUSTER-TOKEN` is the value you discovered earlier.
   - `CLUSTER-NAME` is a unique name of your choice.

   If there are resources to view on the `View` cluster that hosts Tanzu Application Platform GUI, add an entry to `clusters` for it as well.

   If you would like the Kubernetes client to verify the TLS certificates presented by a cluster's API server, set the following properties for the cluster:

```
skipTLSVerify: false
caData: CLUSTER-CA-CERTIFICATES
```

   Where `CLUSTER-CA-CERTIFICATES` is the value you discovered earlier.

2. Update the `tap` package by running this command:

```
tanzu package installed update tap -n tap-install --values-file tap-values.yaml
```

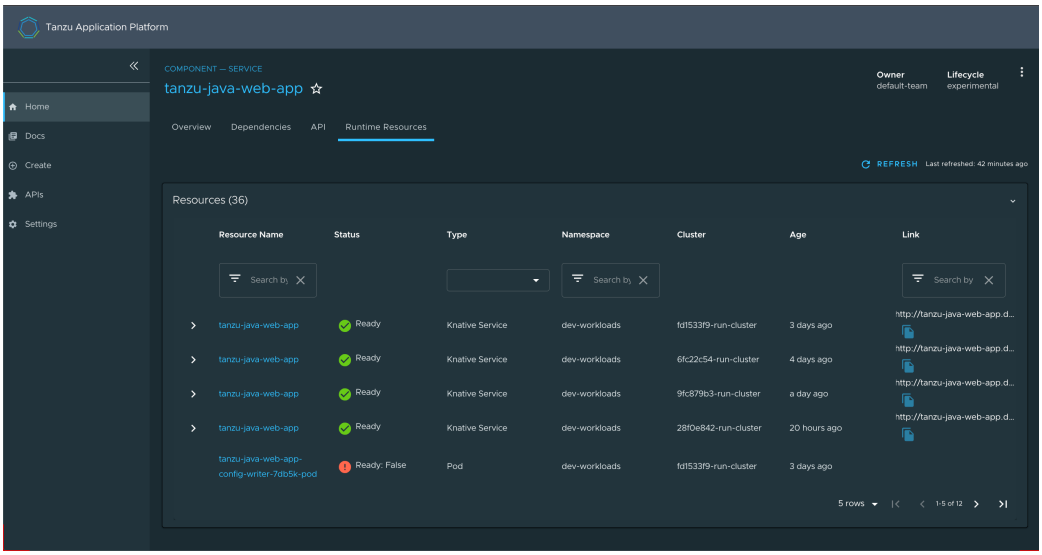
3. Wait a moment for the `tap` and `tap-gui` packages to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
tanzu package installed get all -n tap-install
```

View resources on multiple clusters in the Runtime Resources Visibility plug-in

To view resources on multiple clusters in the Runtime Resources Visibility plug-in:

1. Go to the Runtime Resources Visibility plug-in for a component that is running on multiple clusters.
2. View the multiple resources and their statuses across the clusters.



Set up authentication for Tanzu Application Platform GUI

Tanzu Application Platform GUI (commonly called TAP GUI) extends the current Backstage authentication plug-in so that you can see a login page based on the authentication providers configured at installation. This feature is a work in progress.

Tanzu Application Platform GUI currently supports the following authentication providers:

- Auth0
- Azure
- Bitbucket
- GitHub
- GitLab
- Google
- Okta
- OneLogin

You can also configure a custom OpenID Connect (OIDC) provider.

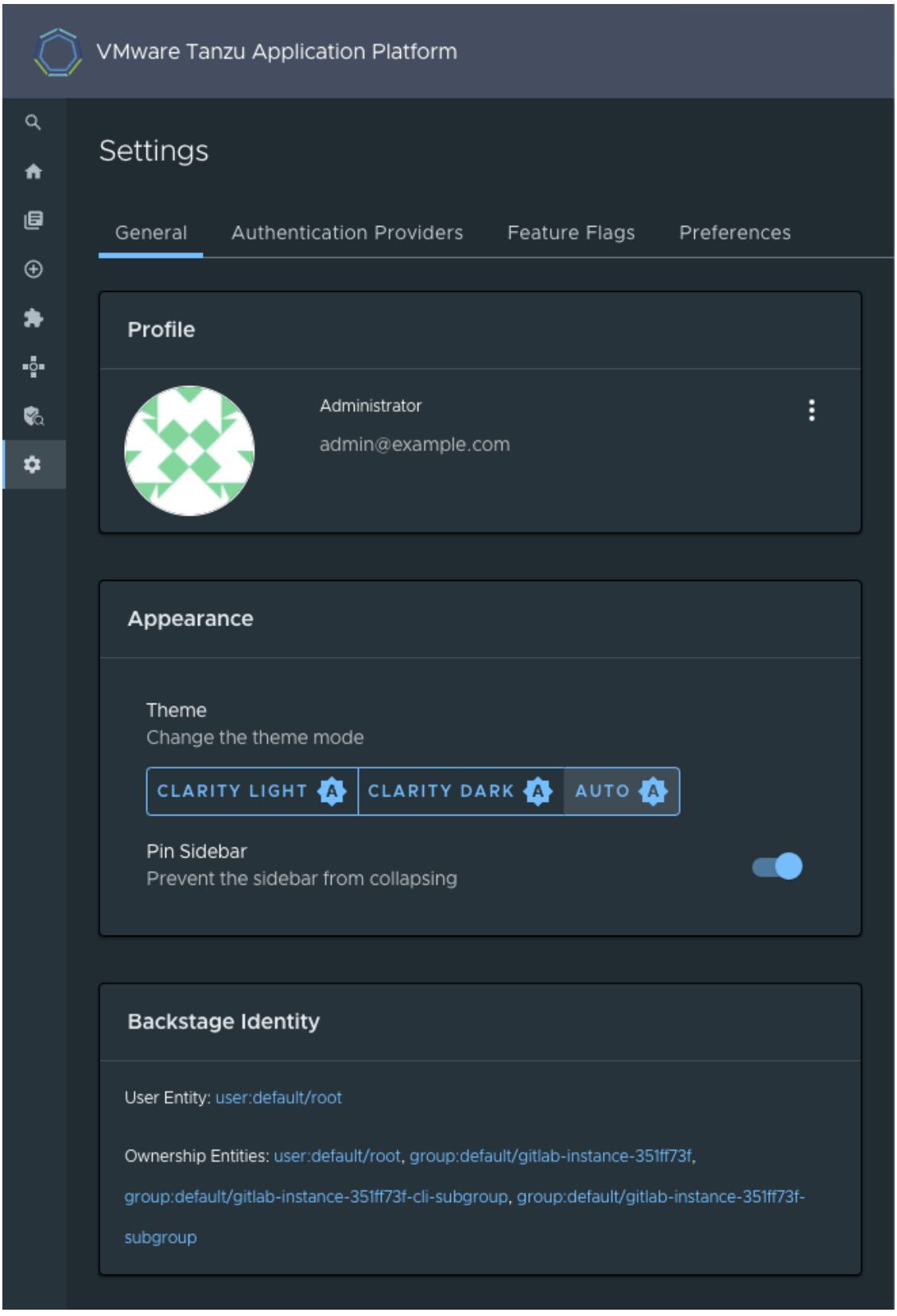
View your Backstage Identity

A Backstage identity is defined as a combination of:

- The user reference: each entity in the catalog is uniquely identified by the triplet of its kind
- A namespace
- A name

For example, the user Jane can be assigned to the user entity `user:default/jane` and an ownership reference, which is used to determine what that user owns. Jane (`user:default/jane`) might have the ownership references `user:default/jane`, `group:default/team-a`, and `group:default/admins`. This would mean that Jane belongs to those groups and, therefore, owns those references.

To view your current Backstage identity, in the **Settings** section of the left side navigation pane click the **General** tab.



Configure an authentication provider

Configure a supported authentication provider or a custom OIDC provider:

- To configure a supported authentication provider, see the Backstage authentication documentation.

- To configure a custom OIDC provider, edit your `tap-values.yaml` file or your custom configuration file to include an OIDC authentication provider. Configure the OIDC provider with your OAuth App values. For example:

```
shared:
  ingress_domain: "INGRESS-DOMAIN"

# ... any existing values

tap_gui:
  # ... any other TAP GUI values
  app_config:
    auth:
      environment: development
      session:
        secret: custom session secret
      providers:
        oidc:
          development:
            metadataUrl: AUTH-OIDC-METADATA-URL
            clientId: AUTH-OIDC-CLIENT-ID
            clientSecret: AUTH-OIDC-CLIENT-SECRET
            tokenSignedResponseAlg: AUTH-OIDC-TOKEN-SIGNED-RESPONSE-ALG # defau
lt='RS256'
            scope: AUTH-OIDC-SCOPE # default='openid profile email'
            prompt: auto # default=none (allowed values: auto, none, consent, l
ogin)
```

  Where `AUTH-OIDC-METADATA-URL` is a JSON file with generic OIDC provider configuration. It contains `authorizationUrl` and `tokenUrl`. Tanzu Application Platform GUI reads these values from `metadataUrl`, so you must not specify these values explicitly in the earlier authentication configuration.

  You must also the provide the redirect URI of the Tanzu Application Platform GUI instance to your identity provider. The redirect URI is sometimes called the redirect URL, the callback URL, or the callback URI. The redirect URI takes the following form:

```
SCHEME://tap-gui.INGRESS-DOMAIN/api/auth/oidc/handler/frame
```

  Where:

  - `SCHEME` is the URI scheme, most commonly `http` or `https`
  - `INGRESS-DOMAIN` is the host name you selected for your Tanzu Application Platform GUI instance

  When using `https` and `example.com` as examples for the two placeholders respectively, the redirect URI reads as follows:

```
https://tap-gui.example.com/api/auth/oidc/handler/frame
```

  For more information, see this example in GitHub.

- (Optional) Configure offline access scope for the OIDC provider by adding the `scope` parameter `offline_access` to either `tap-values.yaml` or your custom configuration file. For example:

```
auth:
  providers:
    oidc:
      development:
```

```
        ... # auth configs
        scope: 'openid profile email offline_access'
```

By default, `scope` is not configured to provide persistence to user login sessions, such as in the case of a page refresh. Not all identity providers support the `offline_access` scope. For more information, see your identity provider documentation.

(Optional) Allow guest access

Enable guest access with other providers by adding the following flag under your authentication configuration:

```
auth:
  allowGuestAccess: true
```

(Optional) Customize the login page

Change the card's title or description for a specific provider with the following configuration:

```
auth:
  environment: development
  providers:
    ... # auth providers config
  loginPage:
    github:
      title: Github Login
      message: Enter with your GitHub account
```

For a provider to appear on the login page, ensure that it is properly configured under the `auth.providers` section of your values file.

View resources on remote clusters

You can control the access to Kubernetes runtime resources on Tanzu Application Platform GUI (commonly called TAP GUI) based on user roles and permissions for each of the visible remote clusters.

Caution

Setting up role-based access control (RBAC) might impact the user's ability to view workloads in the Security Analysis GUI and the Workloads table of the Supply Chain Choreographer plug-in GUI.

RBAC is currently supported for the following Kubernetes cluster providers:

- EKS (Elastic Kubernetes Service) on AWS
- GKE (Google Kubernetes Engine) on GCP

Support for other Kubernetes providers is planned for future releases of Tanzu Application Platform.

Tanzu Application Platform GUI is designed under the assumption that the roles and permissions for the Kubernetes clusters are already defined and that the users are already assigned to their roles. For information about assigning roles and permissions to users, see Assigning roles and permissions on Kubernetes clusters.

Adding access-controlled visibility for a remote cluster is similar to Setting up unrestricted remote cluster visibility.

The steps are:

1. Set up the OIDC provider
2. Configure the Kubernetes cluster with the OIDC provider
3. Configure the Tanzu Application Platform GUI to view the remote cluster
4. Upgrade the Tanzu Application Platform GUI package

After following these steps, you can view your runtime resources on a remote cluster in Tanzu Application Platform GUI. For more information, see View runtime resources on remote clusters.

View resources on remote clusters

You can control the access to Kubernetes runtime resources on Tanzu Application Platform GUI (commonly called TAP GUI) based on user roles and permissions for each of the visible remote clusters.

Caution

Setting up role-based access control (RBAC) might impact the user's ability to view workloads in the Security Analysis GUI and the Workloads table of the Supply Chain Choreographer plug-in GUI.

RBAC is currently supported for the following Kubernetes cluster providers:

- EKS (Elastic Kubernetes Service) on AWS
- GKE (Google Kubernetes Engine) on GCP

Support for other Kubernetes providers is planned for future releases of Tanzu Application Platform.

Tanzu Application Platform GUI is designed under the assumption that the roles and permissions for the Kubernetes clusters are already defined and that the users are already assigned to their roles. For information about assigning roles and permissions to users, see Assigning roles and permissions on Kubernetes clusters.

Adding access-controlled visibility for a remote cluster is similar to Setting up unrestricted remote cluster visibility.

The steps are:

1. Set up the OIDC provider
2. Configure the Kubernetes cluster with the OIDC provider
3. Configure the Tanzu Application Platform GUI to view the remote cluster
4. Upgrade the Tanzu Application Platform GUI package

After following these steps, you can view your runtime resources on a remote cluster in Tanzu Application Platform GUI. For more information, see View runtime resources on remote clusters.

View resources on remote EKS clusters

This topic tells you how to view your runtime resources on a remote EKS cluster in Tanzu Application Platform GUI (commonly called TAP GUI). For more information, see View runtime resources on remote clusters.

Set up the OIDC provider

You must set up the OIDC provider to enable RBAC visibility of remote EKS clusters. You can see the list of supported OIDC providers in Setting up a Tanzu Application Platform GUI authentication provider.

Tanzu Application Platform GUI supports multiple OIDC providers. Auth0 is used here as an example.

1. Log in to the Auth0 dashboard.
2. Go to **Applications**.
3. Create an application of the type `Single Page Web Application` named `TAP-GUI` or a name of your choice.
4. Click the **Settings** tab.
5. Under **Application URIs > Allowed Callback URLs**, add

   ```
   https://tap-gui.INGRESS-DOMAIN/api/auth/auth0/handler/frame
   ```

   Where `INGRESS-DOMAIN` is the domain you chose for your Tanzu Application Platform GUI in Installing the Tanzu Application Platform package and profiles.
6. Click **Save Changes**.

After creating an application with your OIDC provider, you receive the following credentials for setting up RBAC for your remote cluster:

- **Domain**, which is used as `ISSUER-URL` in the following sections (`AUTH0_DOMAIN` for Auth0)
- **Client ID**, which is used as `CLIENT-ID` in the following sections
- **Client Secret**, which is used as `CLIENT-SECRET` in the following sections

For more information, see Auth0 Setup Walkthrough in the Backstage documentation. To configure other OIDC providers, see Authentication in Backstage in the Backstage documentation.

Configure the Kubernetes cluster with the OIDC provider

To configure the cluster with the OIDC provider's credentials:

1. Create a file with the following content and name it `rbac-setup.yaml`. This content applies to EKS clusters.

   ```
   apiVersion: eksctl.io/v1alpha5
   kind: ClusterConfig
   metadata:
     name: "CLUSTER-NAME"
     region: "AWS-REGION"
   identityProviders:
     - name: auth0
       type: oidc
       issuerUrl: "ISSUER-URL"
       clientId: "CLIENT-ID"
       usernameClaim: email
   ```

   Where:

   - `CLUSTER-NAME` is the cluster name for your EKS cluster as an AWS identifier
   - `AWS-REGION` is the AWS region of the EKS cluster
   - `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider
   - `ISSUER-URL` is the Issuer URL you obtained while setting up the OIDC provider. For Auth0, this is `https://${AUTH0_DOMAIN}/`.

2. Using `eksctl`, run:

```
eksctl associate identityprovider -f rbac-setup.yaml
```

3. Verify that the association of the OIDC provider with the EKS cluster was successful by running:

```
eksctl get identityprovider --cluster CLUSTER-NAME
```

Where `CLUSTER-NAME` is the cluster name for your EKS cluster as an AWS identifier

Verify that the output shows `ACTIVE` in the `STATUS` column.

Configure the Tanzu Application Platform GUI

Configure visibility of the remote cluster in Tanzu Application Platform GUI:

1. Obtain your cluster's URL by running:

```
CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.s
erver}')

echo CLUSTER-URL: $CLUSTER_URL
```

This command returns the URL of the first configured cluster in your `kubeconfig` file. To view other clusters one by one, edit the number in `.clusters[0].cluster.server` or edit the command to view all the configured clusters.

2. Ensure you have an `auth` section in the `app_config` section that Tanzu Application Platform GUI uses. In the example for Auth0, copy this YAML content into `tap-values.yaml`:

```
auth:
  environment: development
  providers:
    auth0:
      development:
        clientId: "CLIENT-ID"
        clientSecret: "CLIENT-SECRET"
        domain: "ISSUER-URL"
```

Where:

- `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider.
- `CLIENT-SECRET` is the Client Secret you obtained while setting up the OIDC provider.
- `ISSUER-URL` is the Issuer URL you obtained while setting up the OIDC provider. For Auth0, it is only `AUTH0_DOMAIN`.

3. Add a `kubernetes` section to the `app_config` section that Tanzu Application Platform GUI uses. This section must have an entry for each cluster that has resources to view. To do so, copy this YAML content into `tap-values.yaml`:

```
kubernetes:
  serviceLocatorMethod:
    type: 'multiTenant'
  clusterLocatorMethods:
    - type: 'config'
      clusters:
        - name: "CLUSTER-NAME-UNCONSTRAINED"
          url: "CLUSTER-URL"
          authProvider: oidc
```

```
          oidcTokenProvider: auth0
          skipTLSVerify: true
          skipMetricsLookup: true
```

Where:

- `CLUSTER-NAME-UNCONSTRAINED` is the cluster name of your choice for your EKS cluster
- `CLUSTER-URL` is the URL for the remote cluster you are connecting to Tanzu Application Platform GUI. You obtained this earlier in the procedure.

If there are any other clusters that you want to make visible in Tanzu Application Platform GUI, add their entries to `clusters` as well.

Upgrade the Tanzu Application Platform GUI package

After the new configuration file is ready, update the `tap` package:

1. Run:

   ```
   tanzu package installed update tap --values-file tap-values.yaml
   ```

2. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

   ```
   tanzu package installed get tap-gui -n tap-install
   ```

View resources on remote GKE clusters

This topic tells you about two supported options to add access-controlled visibility for a remote GKE cluster:

- Leverage an external OIDC provider
- Leveraging Google's OIDC provider

After the authorization is enabled, you can view your runtime resources on a remote cluster in Tanzu Application Platform GUI. For more information, see View runtime resources on remote clusters.

Leverage an external OIDC provider

To leverage an external OIDC provider, such as Auth0:

1. Set up the OIDC provider
2. Configure the GKE cluster with the OIDC provider
3. Configure Tanzu Application Platform GUI to view the remote GKE cluster
4. Upgrade `tap-gui` package

Set up the OIDC provider

You must set up the OIDC provider to enable RBAC visibility of remote clusters. You can see the list of supported OIDC providers in Setting up a Tanzu Application Platform GUI authentication provider.

Tanzu Application Platform GUI supports multiple OIDC providers. Auth0 is used here as an example.

1. Log in to the Auth0 dashboard.

2. Go to **Applications**.
3. Create an application of the type `Single Page Web Application` named `TAP-GUI` or a name of your choice.
4. Click the **Settings** tab.
5. Under **Application URIs** > **Allowed Callback URLs**, add

   ```
   https://tap-gui.INGRESS-DOMAIN/api/auth/auth0/handler/frame
   ```

   Where `INGRESS-DOMAIN` is the domain you chose for your Tanzu Application Platform GUI in Installing the Tanzu Application Platform package and profiles.
6. Click **Save Changes**.

After creating an application with your OIDC provider, you receive the following credentials for setting up RBAC for your remote cluster:

- **Domain**, which is used as `issuerURL` in the following sections
- **Client ID**, which is used as `CLIENT-ID` in the following sections
- **Client Secret**, which is used as `CLIENT-SECRET` in the following sections

For more information, see Auth0 Setup Walkthrough in the Backstage documentation. To configure other OIDC providers, see Authentication in Backstage in the Backstage documentation.

Configure the GKE cluster with the OIDC provider

Add redirect configuration on the OIDC side by following the Google Cloud documentation.

Configure visibility of the remote cluster

Configure visibility of the remote cluster in Tanzu Application Platform GUI:

1. Obtain your cluster's URL by running:

   ```
   CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.s
   erver}')

   echo CLUSTER-URL: $CLUSTER_URL
   ```

   This command returns the URL of the first configured cluster in your `kubeconfig` file. To view other clusters one by one, edit the number in `.clusters[0].cluster.server` or edit the command to view all the configured clusters.
2. Ensure you have an `auth` section in the `app_config` section that Tanzu Application Platform GUI uses. In the example for Auth0, copy this YAML content into `tap-values.yaml`:

   ```
   auth:
     environment: development
     providers:
       auth0:
         development:
           clientId: "CLIENT-ID"
           clientSecret: "CLIENT-SECRET"
           domain: "ISSUER-URL"
   ```

   Where:

   - `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider
   - `CLIENT-SECRET` is the Client Secret you obtained while setting up the OIDC provider

- `ISSUER-URL` is the Issuer URL you obtained while setting up the OIDC provider

3. Add a `kubernetes` section to the `app_config` section that Tanzu Application Platform GUI uses. This section must have an entry for each cluster that has resources to view. To do so, copy this YAML content into `tap-values.yaml`:

```
kubernetes:
  serviceLocatorMethod:
    type: 'multiTenant'
  clusterLocatorMethods:
    - type: 'config'
      clusters:
        - name: "CLUSTER-NAME-UNCONSTRAINED"
          url: "CLUSTER-URL"
          authProvider: oidc
          oidcTokenProvider: auth0
          skipTLSVerify: true
          skipMetricsLookup: true
```

Where:

- `CLUSTER-NAME-UNCONSTRAINED` is the cluster name of your choice for your GKE cluster
- `CLUSTER-URL` is the URL for the remote cluster you are connecting to Tanzu Application Platform GUI. You obtained this earlier in the procedure.

If there are any other clusters that you want to make visible in Tanzu Application Platform GUI, add their entries to `clusters` as well.

Update the `tap-gui` package to finish leveraging the external OIDC provider

After the new configuration file is ready, update the `tap-gui` package:

1. Run:

```
tanzu package installed update tap --values-file tap-values.yaml
```

2. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
tanzu package installed get tap-gui -n tap-install
```

Leverage Google's OIDC provider

When leveraging Google's OIDC provider, fewer steps are needed to enable authorization:

1. Add redirect configuration on the OIDC side.
2. Configure the Tanzu Application Platform GUI to view the remote GKE cluster
3. Upgrade the Tanzu Application Platform GUI package

Add redirect configuration on the OIDC side

Add redirect configuration on the OIDC side by following the Google Cloud documentation.

Configure visibility of the remote GKE cluster

Configure visibility of the remote GKE cluster in Tanzu Application Platform GUI:

1. Obtain your cluster's URL by running:

```
CLUSTER_URL=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.s
erver}')

echo CLUSTER-URL: $CLUSTER_URL
```

This command returns the URL of the first configured cluster in your `kubeconfig` file. To view other clusters one by one, edit the number in `.clusters[0].cluster.server` or edit the command to view all the configured clusters.

2. Ensure you have an `auth` section in the `app_config` section that Tanzu Application Platform GUI uses. In the example for Auth0, copy this YAML content into `tap-values.yaml`:

```
auth:
  environment: development
  providers:
    google:
      development:
        clientId: "CLIENT-ID"
        clientSecret: "CLIENT-SECRET"
```

Where:

- `CLIENT-ID` is the Client ID you obtained while setting up the OIDC provider
- `CLIENT-SECRET` is the Client Secret you obtained while setting up the OIDC provider

3. Add a `kubernetes` section to the `app_config` section that Tanzu Application Platform GUI uses. This section must have an entry for each cluster that has resources to view. To do so, copy this YAML content into `tap-values.yaml`:

```
kubernetes:
  clusterLocatorMethods:
    - type: 'config'
      clusters:
        - name: "CLUSTER-NAME-UNCONSTRAINED"
          url: "CLUSTER-URL"
          authProvider: google
          caData: "CA-DATA"
```

Where:

- `CLUSTER-NAME-UNCONSTRAINED` is the cluster name of your choice for your GKE cluster.
- `CLUSTER-URL` is the URL for the remote cluster you are connecting to Tanzu Application Platform GUI. You obtained this earlier in the procedure.
- `CA-DATA` is the CA certificate data.

If there are any other clusters that you want to make visible in Tanzu Application Platform GUI, add their entries to `clusters` as well.

Update the `tap-gui` package to finish leveraging the Google OIDC provider

After the new configuration file is ready, update the `tap-gui` package:

1. Run:

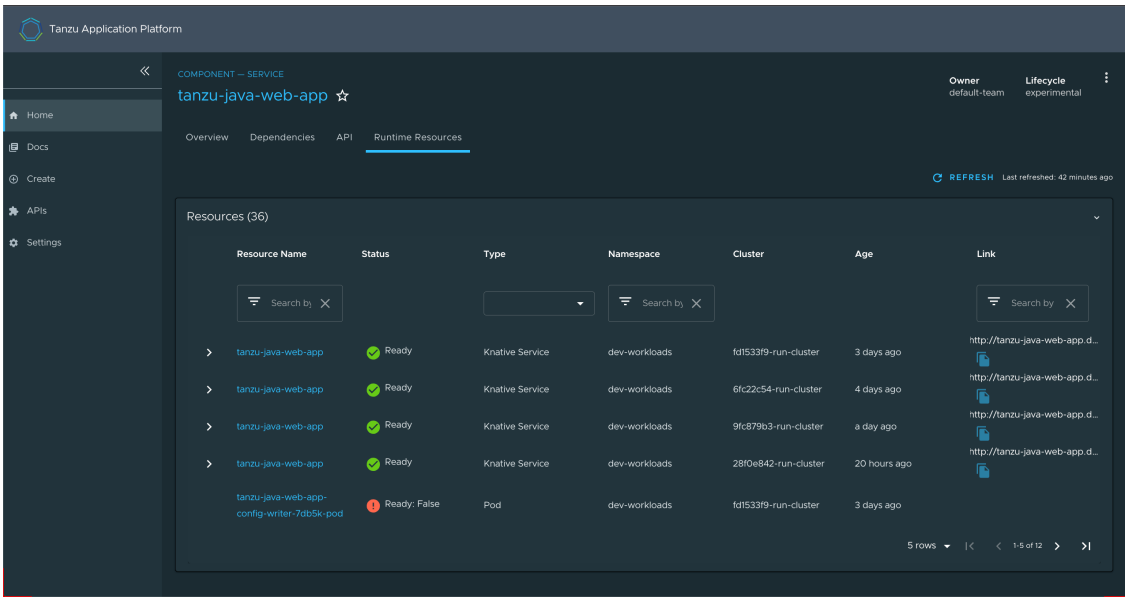
```
tanzu package installed update tap --values-file tap-values.yaml
```

2. Wait a moment for the `tap-gui` package to update and then verify that `STATUS` is `Reconcile succeeded` by running:

```
tanzu package installed get tap-gui -n tap-install
```

View runtime resources on authorization-enabled clusters

To visualize runtime resources on authorization-enabled clusters in Tanzu Application Platform GUI (commonly called TAP GUI), proceed to the software catalog component of choice and click the **Runtime Resources** tab on top of the ribbon.



After you click **Runtime Resources**, Tanzu Application Platform GUI uses your credentials to query the clusters for the respective runtime resources. The system verifies that you are authenticated with the OIDC providers configured for the remote clusters. If you are not authenticated, the system prompts you for your OIDC credentials.

Remote clusters that are not restricted by authorization are visible by using the general Service Account of Tanzu Application Platform GUI. It is not restricted for users. For more information about how to set up unrestricted remote cluster visibility, see Viewing resources on multiple clusters in Tanzu Application Platform GUI.

The type of query to the remote cluster depends on the definition of the software catalog component. In Tanzu Application Platform GUI, there are globally-scoped components and namespace-scoped components.

This property of the component affects runtime resource visibility, depending on your permissions on a specific cluster.

If your permissions on the authorization-enabled cluster are limited to specific namespaces, you do not have visibility into runtime resources of globally-scoped components.

You need cluster-scoped access to have visibility into runtime resources of globally-scoped components.

Globally-scoped components

For globally-scoped components, when you access **Runtime Resources** Tanzu Application Platform GUI queries all Kubernetes namespaces for runtime resources that have a matching `kubernetes-label-selector`, usually with a `part-Of` prefix.

For example, `demo-component-a` does not have a `backstage.io/kubernetes-namespace` in the `metadata.annotations` section. This makes it a globally-scoped component. See the following

example YAML.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-component-a
  description: Demo Component A
  tags:
    - java
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=component-a'
spec:
  type: service
  lifecycle: experimental
  owner: team-a
```

Namespace-scoped components

If a component is namespace-scoped, when you access **Runtime Resources** Tanzu Application Platform GUI queries only the associated Kubernetes namespace for each remote cluster that is visible to Tanzu Application Platform GUI.

To make a component namespace-scoped, pass the following annotation to the definition YAML file of the component:

```
annotations:
  'backstage.io/kubernetes-namespace': NAMESPACE-NAME
```

Where `NAMESPACE-NAME` is the Kubernetes namespace you want to associate your component with.

For example, `demo-component-b` has a `kubernetes-namespace` in the `metadata.annotations` section, which associates it with the `component-b` namespaces on each of the visible clusters. This makes it a namespace-scoped component. See the following example YAML.

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-component-b
  description: Demo Component B
  tags:
    - java
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=component-b'
    'backstage.io/kubernetes-namespace': component-b
spec:
  type: service
  lifecycle: experimental
  owner: team-b
```

When the `kubernetes-namespace` annotation is absent, the component is considered globally-scoped by default. For more information, see Adding Namespace Annotation in the Backstage documentation.

Assign roles and permissions on Kubernetes clusters

This topic gives you an overview of creating roles and permissions on Kubernetes clusters and assigning these roles to users. For more information, see Using RBAC Authorization in the Kubernetes documentation.

The steps to define and assign roles are:

1. Create roles
2. Create users
3. Assign users to their roles

Create roles

To control the access to Kubernetes runtime resources on Tanzu Application Platform GUI based on users' roles and permissions for each of visible remote clusters, VMware recommends two role types:

- Cluster-scoped roles
- Namespace-scoped roles

Cluster-scoped roles

Cluster-scoped roles provide cluster-wide privileges. They enable visibility into runtime resources across all of a cluster's namespaces.

In this example YAML snippet, the `pod-viewer` role enables pod visibility on the cluster:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-viewer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Namespace-scoped roles

Namespace-scoped roles provide privileges that are limited to a certain namespace. They enable visibility into runtime resources inside namespaces.

In this example YAML snippet, the `pod-viewer-app1` role enables pod visibility in the `app1` namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: app1
  name: pod-viewer-app1
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```

Create users

You can create users by running the `kubectl create` command. In this example YAML snippet, the user `john` is defined:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: User
metadata:
  namespace: default
  name: john
```

Assign users to their roles

After the users and role are created, the next step is to bind them together.

To bind a Tanzu Application Platform default role, see Bind a user or group to a default role.

In this example YAML snippet, the user `john` is bound with the `pod-viewer` cluster role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: john-pod-viewer
  namespace: default
subjects:
- kind: User
  name: john
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: pod-viewer
  apiGroup: rbac.authorization.k8s.io
```

In this example YAML snippet, the user `john` is bound with the `pod-viewer-app1` namespace-specific role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: john-pod-viewer-app1
  namespace: app1
subjects:
- kind: User
  name: john
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-viewer-app1
  apiGroup: rbac.authorization.k8s.io
```

To verify the user's permissions, run the `can-i` commands to get a `yes` or `no` answer. To verify that you can list pods in your cluster-wide role, run:

```
kubectl auth can-i get pods --all-namespaces
```

To verify that you can list pods in namespace `app1` in your namespace-specific role, run:

```
kubectl auth can-i get pods --namespace app1
```

Add Tanzu Application Platform GUI integrations

You can integrate Tanzu Application Platform GUI (commonly called TAP GUI) with several Git providers. To use an integration, you must enable it and provide the necessary token or credentials in `tap-values.yaml`.

Add a GitHub provider integration

To add a GitHub provider integration, edit `tap-values.yaml` as in this example:

```
app_config:
  app:
```

```
    baseUrl: http://EXTERNAL-IP:7000
  # Existing tap-values.yaml above
  integrations:
    github: # Other integrations available see NOTE below
      - host: github.com
        token: GITHUB-TOKEN
```

Where:

- `EXTERNAL-IP` is the external IP address.
- `GITHUB-TOKEN` is a valid token generated from your Git infrastructure of choice. Ensure that `GITHUB-TOKEN` has the necessary read permissions for the catalog definition files you extracted from the blank software catalog introduced in the Tanzu Application Platform GUI prerequisites.

Add a Git-based provider integration that isn't GitHub

To enable Tanzu Application Platform GUI to read Git-based non-GitHub repositories containing component information:

1. Add the following YAML to `tap-values.yaml`:

   ```
   app_config:
     # Existing tap-values.yaml above
     backend:
       reading:
         allow:
           - host: "GIT-CATALOG-URL-1"
           - host: "GIT-CATALOG-URL-2" # Including more than one URL is optional
   ```

   Where `GIT-CATALOG-URL-1` and `GIT-CATALOG-URL-2` are URLs in a list of URLs that Tanzu Application Platform GUI can read when registering new components. For example, `git.example.com`. For more information about registering new components, see Adding catalog entities.

2. Adding the YAML from the previous step currently causes the **Accelerators** page to break and not show any accelerators. Provide a value for Application Accelerator as a workaround, as in this example:

   ```
   app_config:
     # Existing tap-values.yaml above
     backend:
       reading:
         allow:
           - host: acc-server.accelerator-system.svc.cluster.local
   ```

Add a non-Git provider integration

To add an integration for a provider that isn't associated with GitHub, see the Backstage documentation.

Update the package profile

After changing `tap-values.yaml`, update the package profile by running:

```
tanzu package installed update  tap --package tap.tanzu.vmware.com --version VERSION-N
UMBER \
--values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is the Tanzu Application Platform version. For example, `1.5.12`.

For example:

```
$ tanzu package installed update  tap --package tap.tanzu.vmware.com --version \
1.5.12 --values-file tap-values.yaml -n tap-install
| Updating package 'tap'
| Getting package install for 'tap'
| Getting package metadata for 'tap.tanzu.vmware.com'
| Updating secret 'tap-tap-install-values'
| Updating package install for 'tap'
/ Waiting for 'PackageInstall' reconciliation for 'tap'


Updated package install 'tap' in namespace 'tap-install'
```

Configure the Tanzu Application Platform GUI database

The Tanzu Application Platform GUI (commonly called TAP GUI) catalog gives you two approaches for storing catalog information:

- **In-memory database:**

  The default option uses an in-memory database and is suitable for test and development scenarios only. The in-memory database reads the catalog data from Git URLs that you write in `tap-values.yaml`.

  This data is temporary. Any operations that cause the `server` pod in the `tap-gui` namespace to be re-created also cause this data to be rebuilt from the Git location.

  This can cause issues when you manually register entities by using the UI because they only exist in the database and are lost when that in-memory database is rebuilt. If you choose this method, you lose all user preferences and any manually registered entities when the Tanzu Application Platform GUI server pod is re-created.

- **PostgreSQL database:**

  For production use-cases, use a PostgreSQL database that exists outside the Tanzu Application Platform packaging. The PostgreSQL database stores all the catalog data persistently both from the Git locations and the UI manual entity registrations.

For production or general-purpose use-cases, a PostgreSQL database is recommended.

Configure a PostgreSQL database

See the following sections for configuring Tanzu Application Platform GUI to use a PostgreSQL database.

Edit `tap-values.yaml`

Apply the following values in `tap-values.yaml`:

```
# ... existing tap-values.yaml above
tap_gui:
  # ... existing tap_gui values
  app_config:
    backend:
      database:
        client: pg
        connection:
          host: PG-SQL-HOSTNAME
          port: 5432
```

```
          user: PG-SQL-USERNAME
          password: PG-SQL-PASSWORD
          ssl: {rejectUnauthorized: false} # Set to true if using SSL
```

Where:

- `PG-SQL-HOSTNAME` is the host name of your PostgreSQL database
- `PG-SQL-USERNAME` is the user name of your PostgreSQL database
- `PG-SQL-PASSWORD` is the password of your PostgreSQL database

(Optional) Configure extra parameters

Beyond the minimum configuration options needed to make Tanzu Application Platform GUI work with the `pg` driver, there are many more configuration options for other purposes. For example, you can restrict Tanzu Application Platform GUI to a single database. For more information about this restriction, see the Backstage documentation.

By default, Tanzu Application Platform GUI creates a database for each plug-in, but you can configure it to divide plug-ins based on different PostgreSQL schemas and use a single specified database.

See the following example of extra configuration parameters:

```
# ... existing tap-values.yaml above
tap_gui:
  # ... existing tap_gui values
  app_config:
    backend:
      # ... other backend details
      database:
        client: pg

        # This parameter tells Tanzu Application Platform GUI to put plug-ins in their
own schema instead
        # of their own database.
        # default: database
        pluginDivisionMode: schema

        connection:
          # ... other connection details
          database: PG-SQL-DATABASE
```

Where `PG-SQL-DATABASE` is the database name for Tanzu Application Platform GUI to use

For the complete list of these configuration options, see the node-postgres documentation.

Update the package profile

You can apply your new configuration by updating Tanzu Application Platform with your modified values. Doing so updates Tanzu Application Platform GUI because it belongs to Tanzu Application Platform.

To apply your new configuration, run:

```
tanzu package installed update  tap --package tap.tanzu.vmware.com --version VERSION-N
UMBER --values-file tap-values.yaml -n tap-install
```

Where `VERSION-NUMBER` is your Tanzu Application Platform version. For example, `1.5.12`.

For example:

```
$ tanzu package installed update  tap --package tap.tanzu.vmware.com --version {{ var
s.tap_version }} --values-file tap-values.yaml -n tap-install
| Updating package 'tap'
| Getting package install for 'tap'
| Getting package metadata for 'tap.tanzu.vmware.com'
| Updating secret 'tap-tap-install-values'
| Updating package install for 'tap'
/ Waiting for 'PackageInstall' reconciliation for 'tap'


Updated package install 'tap' in namespace 'tap-install'
```

Generate and publish TechDocs

This topic tells you how to generate and publish TechDocs for catalogs as part of Tanzu Application Platform GUI (commonly called TAP GUI). For more information about TechDocs, see the Backstage.io documentation.

Create an Amazon S3 bucket

To create an Amazon S3 bucket:

1. Go to Amazon S3.
2. Click **Create bucket**.
3. Give the bucket a name.
4. Select the AWS region.
5. Keep **Block all public access** checked.
6. Click **Create bucket**.

Configure Amazon S3 access

The TechDocs are published to the S3 bucket that was recently created. You need an AWS user's access key to read from the bucket when viewing TechDocs.

Create an AWS IAM user group

To create an AWS IAM User Group:

1. Click **Create Group**.
2. Give the group a name.
3. Click **Create Group**.
4. Click the new group and navigate to **Permissions**.
5. Click **Add permissions** and click **Create Inline Policy**.
6. Click the **JSON** tab and replace contents with this JSON replacing `BUCKET-NAME` with the bucket name.

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Sid": "ReadTechDocs",
          "Effect": "Allow",
          "Action": [
```

```
                "s3:ListBucket",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::BUCKET-NAME",
                "arn:aws:s3:::BUCKET-NAME/*"
            ]
        }
    ]
}
```

7. Click **Review policy**.
8. Give the policy a name and click **Create policy**.

Create an AWS IAM user

To create an AWS IAM User to add to this group:

1. Click **Add users**.
2. Give the user a name.
3. Verify **Access key - Programmatic access** and click **Next: Permissions**.
4. Verify the IAM Group to add the user to and click **Next: Tags**.
5. Click **Next: Review** then click **Create user**.
6. Record the **Access key ID** (`AWS_READONLY_ACCESS_KEY_ID`) and the **Secret access key** (`AWS_READONLY_SECRET_ACCESS_KEY`) and click **Close**.

Find the catalog locations and their entities' namespace, kind, and name

TechDocs are generated for catalogs that have Markdown source files for TechDocs. To find the catalog locations and their entities' namespace, kind, and name:

1. The catalogs appearing in Tanzu Application Platform GUI are listed in the config values under `app_config.catalog.locations`.
2. For a catalog, clone the catalog's repository to the local file system.
3. Find the `mkdocs.yml` that is at the root of the catalog. There is a YAML file describing the catalog at the same level called `catalog-info.yaml`.
4. Record the values for `namespace`, `kind`, and `metadata.name`, and the directory path containing the YAML file.
5. Record the `spec.targets` in that file.
6. Find the namespace, kind, or name for each of the targets:
    1. Go to the target's YAML file.
    2. The `namespace` value is the value of `namespace`. If it is not specified, it has the value `default`.
    3. The `kind` value is the value of `kind`.
    4. The `name` value is the value of `metadata.name`.
    5. Record the directory path containing the YAML file.

Use the TechDocs CLI to generate and publish TechDocs

VMware uses `npx` to run the TechDocs CLI, which requires `Node.js` and `npm`. To generate and publish TechDocs by using the TechDocs CLI:

1. Download and install Node.js and npm.
2. Install `npx` by running:

   ```
   npm install -g npx
   ```

3. Generate the TechDocs for the root of the catalog by running:

   ```
   npx @techdocs/cli generate --source-dir DIRECTORY-CONTAINING-THE-ROOT-YAML-FILE
   --output-dir ./site
   ```

   Note

   This creates a temporary `site` directory in your current working directory that contains the generated TechDocs files.

4. Review the contents of the `site` directory to verify the TechDocs were generated.
5. Set environment variables for authenticating with Amazon S3 with an account that has read/write access:

   ```
   export AWS_ACCESS_KEY_ID=AWS-ACCESS-KEY-ID
   export AWS_SECRET_ACCESS_KEY=AWS-SECRET-ACCESS-KEY
   export AWS_REGION=AWS-REGION
   ```

6. Publish the TechDocs for the root of the catalog to the Amazon S3 bucket you created earlier by running:

   ```
   npx @techdocs/cli publish --publisher-type awsS3 --storage-name BUCKET-NAME --e
   ntity \
   NAMESPACE/KIND/NAME --directory ./site
   ```

   Where `NAMESPACE/KIND/NAME` are the values for `namespace`, `kind`, and `metadata.name` you recorded earlier. For example, `default/location/yelb-catalog-info`.

7. For each of the `spec.targets` found earlier, repeat the `generate` and `publish` commands.

   Note

   The `generate` command erases the contents of the `site` directory before creating new TechDocs files. Therefore, the `publish` command must follow the `generate` command for each target.

Update the `techdocs` section in `app-config.yaml` to point to the Amazon S3 bucket

Update the config values you used during installation to point to the Amazon S3 bucket that has the published TechDocs files:

1. Add or edit the `techdocs` section under `app_config` in the config values with the following YAML, replacing placeholders with the appropriate values.

```
techdocs:
  builder: 'external'
  publisher:
    type: 'awsS3'
    awsS3:
      bucketName: BUCKET-NAME
      accountId: ACCOUNT-ID
      region: AWS-REGION
aws:
  accounts:
    - accountId: ACCOUNT-ID
      accessKeyId: AWS-READONLY-ACCESS-KEY-ID
      secretAccessKey: AWS-READONLY-SECRET-ACCESS-KEY
```

For more information about authentication to an Amazon S3 bucket through an assumed role, see the Backstage documentation.

2. Update your installation from the Tanzu CLI:

Tanzu Application Platform package installation
If you installed Tanzu Application Platform GUI as part of the Tanzu Application Platform package (in other words, if you installed it by running `tanzu package install tap ...`) then run:

```
tanzu package installed update tap \
  --version PACKAGE-VERSION \
  --values-file VALUES-FILE
```

Where `PACKAGE-VERSION` is your package version and `VALUES-FILE` is your values file

Separate package installation
If you installed Tanzu Application Platform GUI as its own package (in other words, if you installed it by running `tanzu package install tap-gui ...`) then run:

```
tanzu package installed update tap-gui \
  --version PACKAGE-VERSION \
  --values-file VALUES-FILE
```

Where `PACKAGE-VERSION` is your package version and `VALUES-FILE` is your values file

3. Verify the status of the update by running:

```
tanzu package installed list
```

4. Go to the **Docs** section of your catalog and view the TechDocs pages to verify the content is loaded from the S3 bucket.

Overview of Tanzu Application Platform GUI plug-ins

Tanzu Application Platform GUI (commonly called TAP GUI) has many pre-integrated plug-ins. You need not configure the plug-ins. To use a plug-in, you must install the relevant Tanzu Application Platform component.

Tanzu Application Platform has the following GUI plug-ins:

- Runtime Resources Visibility
- Application Live View
- Application Accelerator

- API Documentation
- Security Analysis
- Supply Chain Choreographer

Overview of Tanzu Application Platform GUI plug-ins

Tanzu Application Platform GUI (commonly called TAP GUI) has many pre-integrated plug-ins. You need not configure the plug-ins. To use a plug-in, you must install the relevant Tanzu Application Platform component.

Tanzu Application Platform has the following GUI plug-ins:

- Runtime Resources Visibility
- Application Live View
- Application Accelerator
- API Documentation
- Security Analysis
- Supply Chain Choreographer

Runtime resources visibility in Tanzu Application Platform GUI

This topic tells you about runtime resources visibility.

The Runtime Resources Visibility plug-in enables users to visualize their Kubernetes resources associated with their workloads.

Prerequisite

Do one of the following actions to access the Runtime Resources Visibility plug-in:

- Install the Tanzu Application Platform Full or View profile
- Install Tanzu Application Platform without using a profile and then install Tanzu Application Platform GUI separately
- Review the section If you have a metrics server

If you have a metrics server

By default, the Kubernetes API does not attempt to use any metrics servers on your clusters. To access metrics information for a cluster, set `skipMetricsLookup` to `false` for that cluster in the `kubernetes` section of `app-config.yaml`. Example:

```
tap_gui:
  # ... existing configuration
  app_config:
    # ... existing configuration
    kubernetes:
      clusterLocatorMethods:
        - type: 'config'
          clusters:
            - url: https://KUBERNETES-SERVICE-HOST:KUBERNETES-SERVICE-PORT
              name: host
              authProvider: serviceAccount
              serviceAccountToken: KUBERNETES-SERVICE-ACCOUNT-TOKEN
```

```
            skipTLSVerify: true
            skipMetricsLookup: false
```

Where:

- `KUBERNETES-SERVICE-HOST` and `KUBERNETES-SERVICE-PORT` are the URL and ports of your Kubernetes cluster. You can gather these through `kubectl cluster-info`.
- `KUBERNETES-SERVICE-ACCOUNT-TOKEN` is the token from your `tap-gui-token-id`.

You can retrieve this secret's ID by running:

```
kubectl get secrets -n tap-gui
```

and then running

```
kubectl describe secret tap-gui-token-ID
```

Where `ID` is the secret name from the first step.

Caution

If you enable metrics for a cluster but do not have a metrics server running on it, Tanzu Application Platform web interface users see an error notifying them that there is a problem connecting to the back end.

Visualize Workloads on Tanzu Application Platform GUI

In order to view your applications on Tanzu Application Platform GUI, use the following steps:

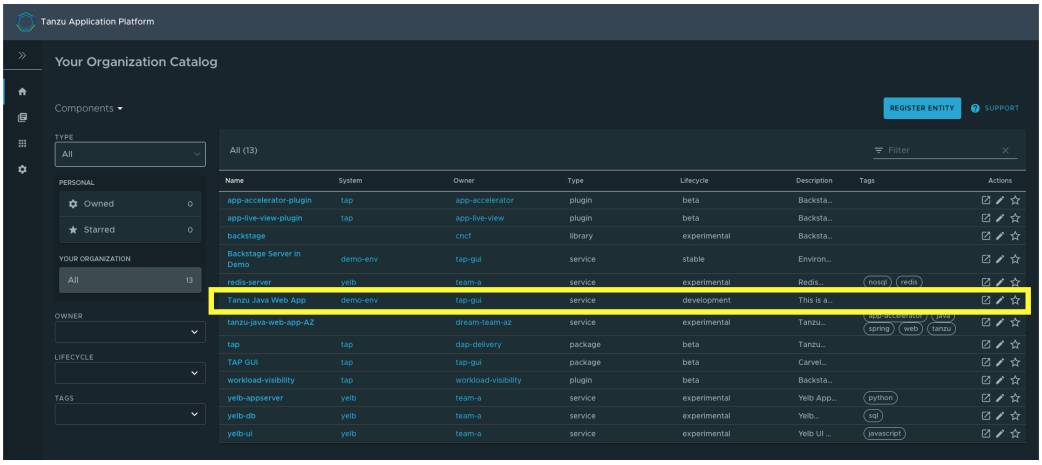
1. Deploy your first application on the Tanzu Application Platform
2. Add your application to Tanzu Application Platform GUI Software Catalog

Navigate to the **Runtime Resources Visibility** screen

You can view the list of running resources and the details of their status, type, namespace, cluster, and public URL if applicable for the resource type.

To view the list of your running resources:

1. Select your component from the Catalog index page.



2. Select the **Runtime Resources** tab.

Resources

Built-in Kubernetes resources in this view are:

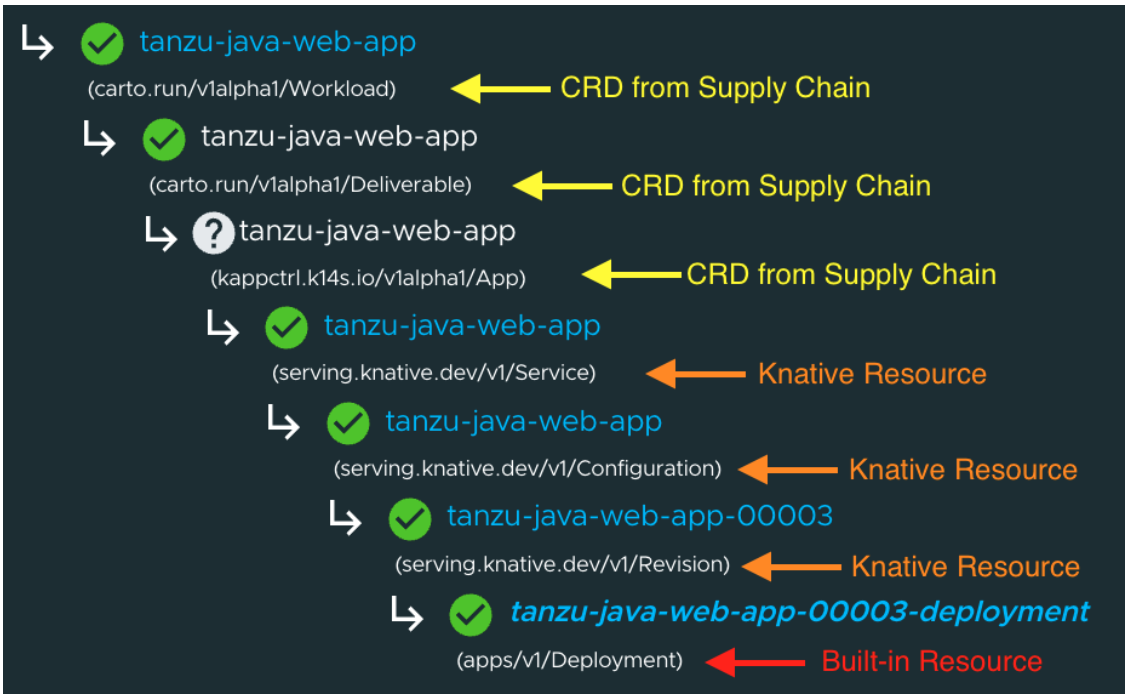
- Services
- Deployments
- ReplicaSets
- Pods
- Jobs
- Cronjobs
- DaemonSets
- ReplicaSets

The Runtime Resource Visibility plug-in also displays CRDs created with the Supply Chain, including:

- Cartographer Workloads
- Knative Services, Configurations, Revisions, and Routes

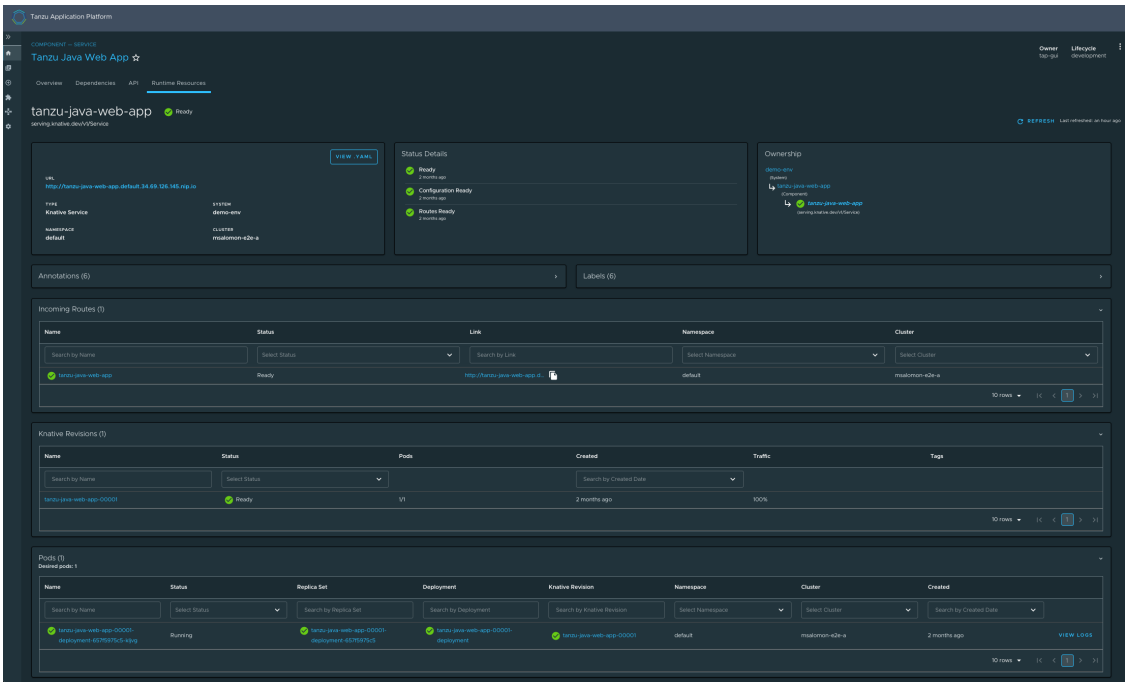
For more information, see Supply Chain Choreographer in Tanzu Application Platform GUI.

CRDs from Supply Chain are associated with Knative Resources, further down the chain, and built-in resources even further down the chain.



Resources details page

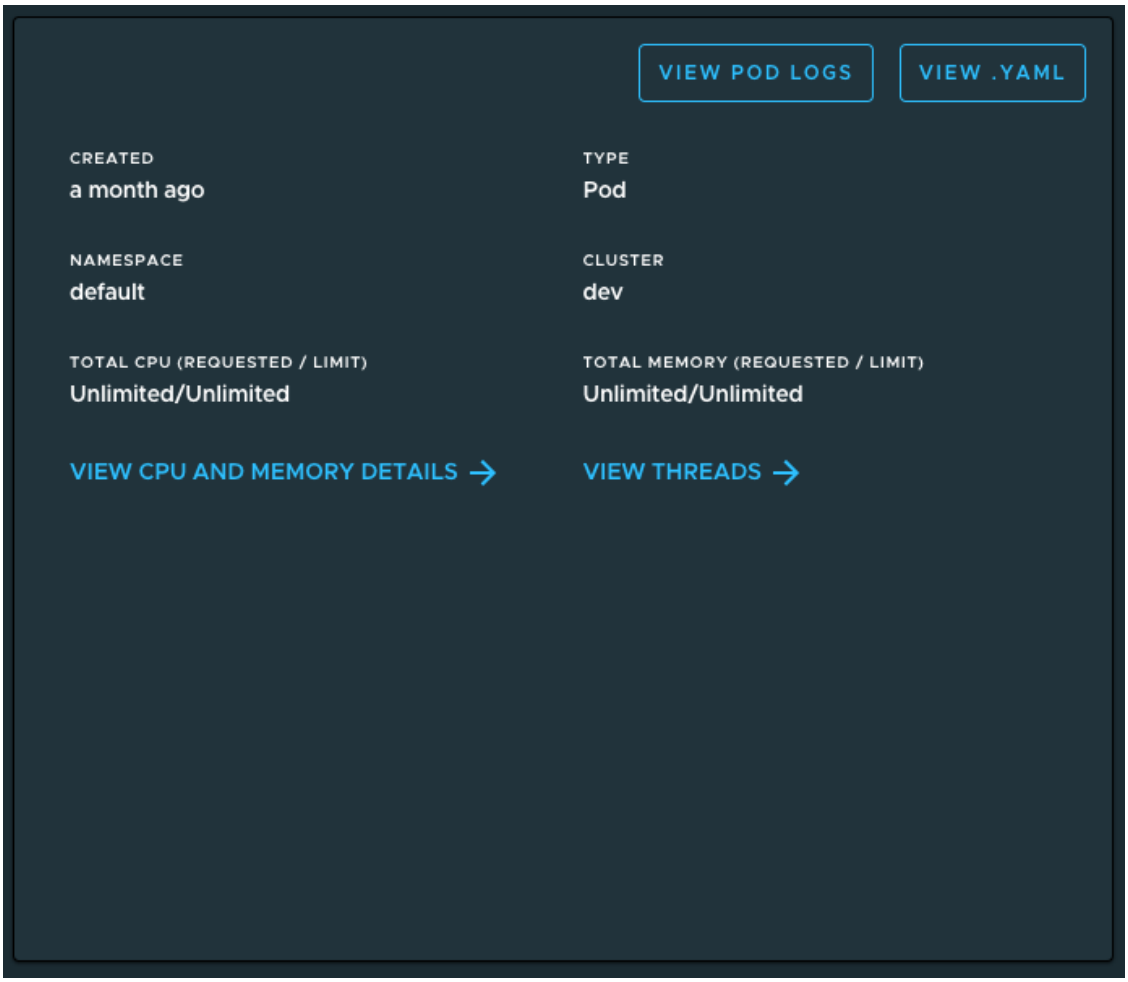
To get more information about a particular workload, select it from the table on the main **Runtime Resources** page to visit a page that provides details about the workload. These details include the workload status, ownership, and resource-specific information.



Overview card

All detail pages provide an overview card with information related to the selected resource. Most of the information feeds from the `metadata` attribute in each object. The following are some attributes that are displayed in the overview card:

- **View Pod Logs** button
- **View .YAML** button
- URL, which is for Knative and Kubernetes service detail pages
- Type
- System
- Namespace
- Cluster



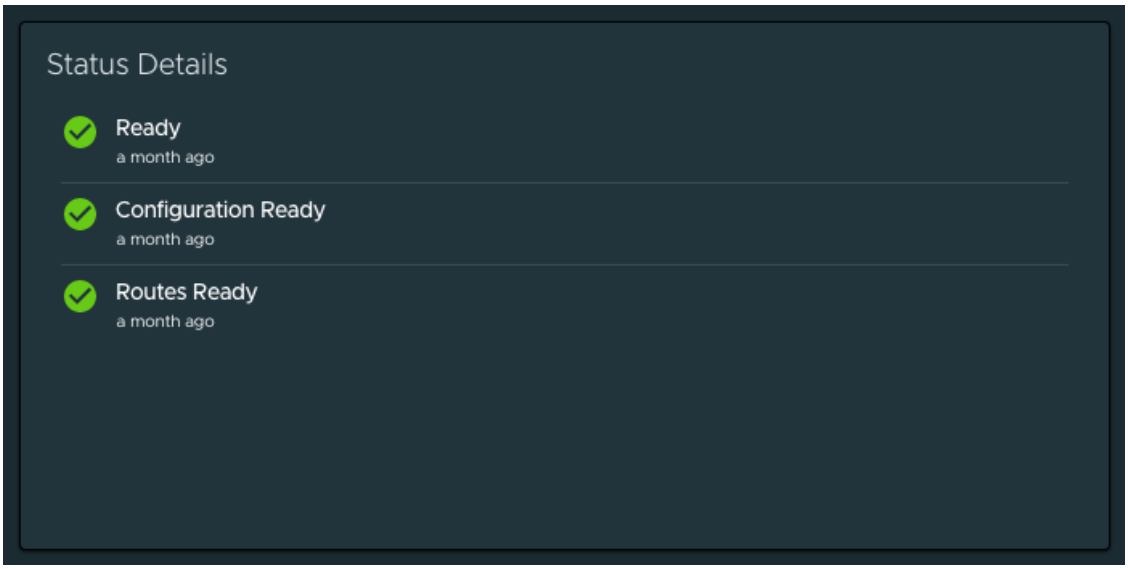
Note

The **VIEW CPU AND MEMORY DETAILS** and **VIEW THREADS** sections are only available for applications supporting Application Live View.

Status card

The status section displays all of the conditions in the resource's attribute `status.conditions`. Not all resources have conditions, and they can vary from one resource to the other.

For more information about object `spec` and `status`, see the Kubernetes documentation.



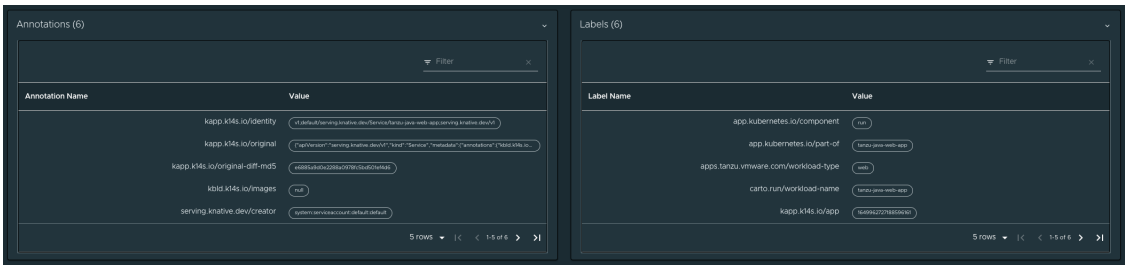
Ownership card

Depending on the resource that you are viewing, the ownership section displays all the resources specified in `metadata.ownerReferences`. You can use this section to navigate between resources.

For more information about owners and dependents, see the Kubernetes documentation.

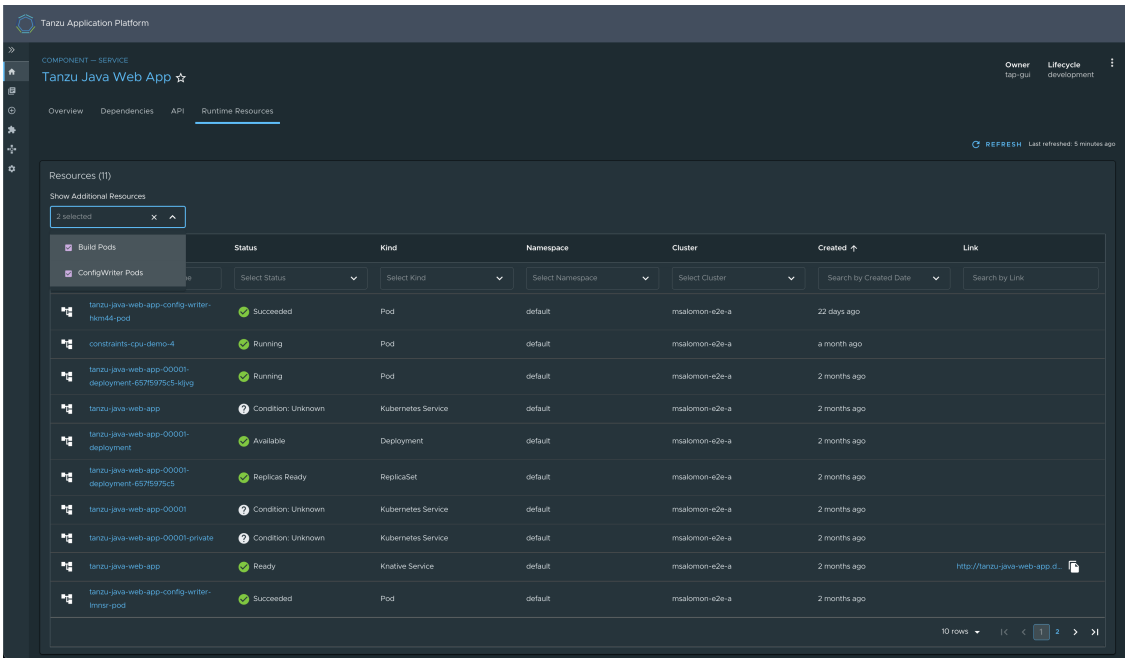
Annotations and Labels

The Annotations and Labels card displays information about `metadata.annotations` and `metadata.labels`.



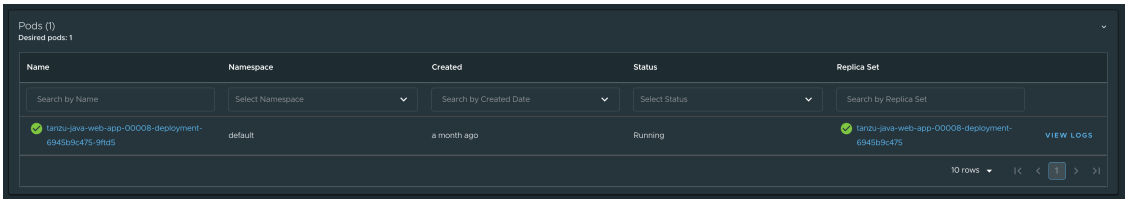
Selecting completed supply chain pods

Completed supply chain pods (build pods and ConfigWriter pods) are hidden by default in the index table. Users can choose to display them from the **Show Additional Resources** drop-down menu above the Resources index table. This drop-down menu is only visible if the resources include Build or ConfigWriter pods.



Navigating to the pod Details page

Users can see the pod table in each resource details page.



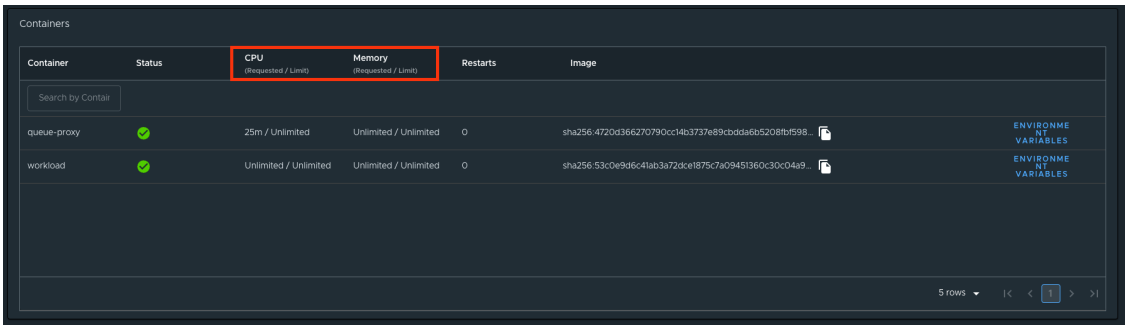
Overview of pod metrics

If you have a metrics server running on your cluster, the overview card displays realtime metrics for pods.

If you do not have a metrics server, the overview card displays the user-configured resource limits on the pod, defined in accordance with the Kubernetes documentation.

For applications built using Spring Boot, you can also monitor the actual real-time resource use using Screenshot of Application Live View for Spring Boot Applications in Tanzu Application Platform GUI..

Metrics and limits are also displayed for each container on a pod details page. If a particular container's current limit conflicts with a namespace-level LimitRange, a small warning indicator is displayed next to the container limit. Most conflicts are due to creating a container before applying a LimitRange.



Pods display the sum of the limits of all their containers. If a limit is not specified for a container, both the container and its pod are deemed to require unlimited resources.

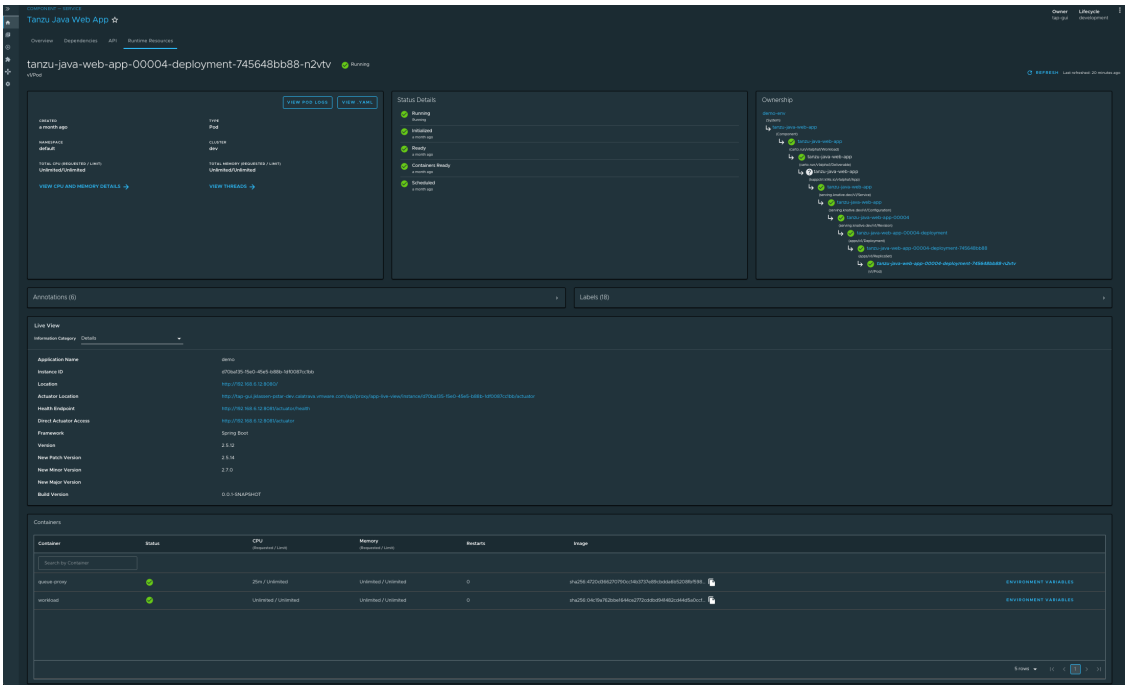
Namespace-level resource limits, such as default memory limits and default CPU limits, are not considered as part of these calculations.

For more information about default memory limits and default CPU limits see the Kubernetes documentation.

These limits apply only for Memory and CPU that a pod or container can use. Kubernetes manages these resource units by using a binary base, which is explained in the Kubernetes documentation.

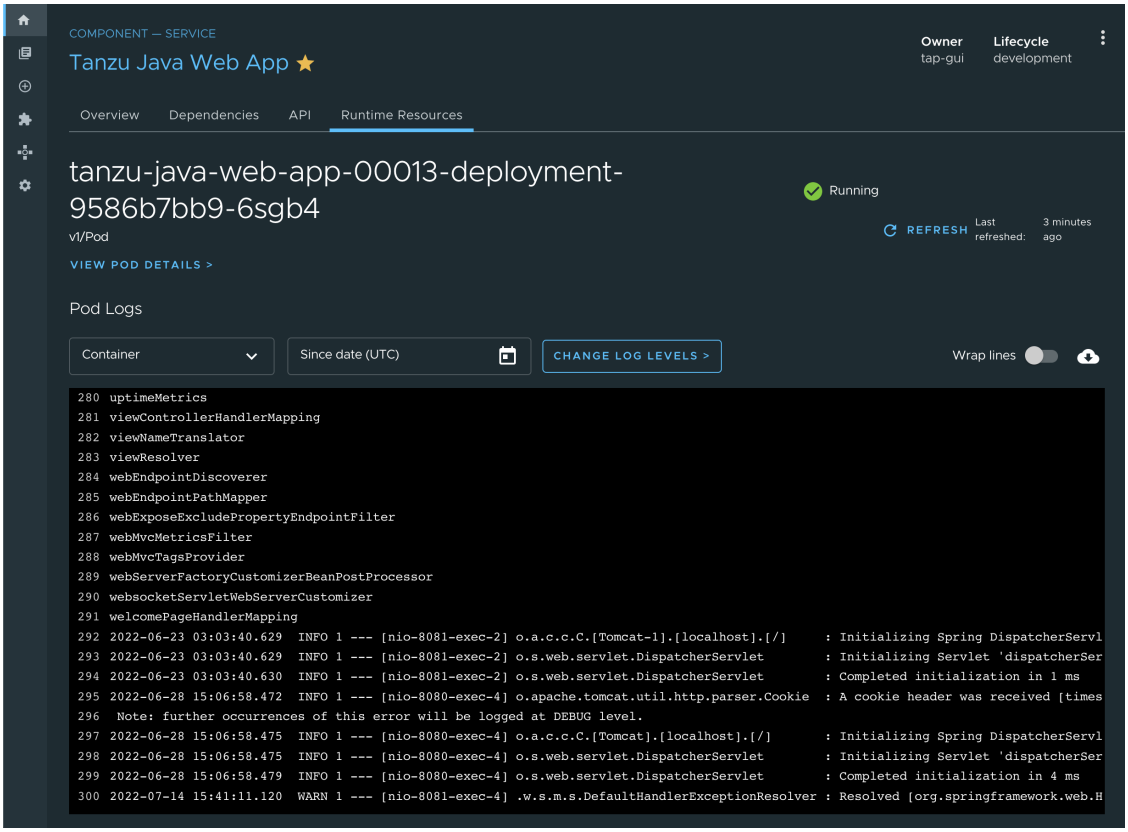
Navigating to Application Live View

To view additional information about your running applications, see the Application Live View section in the **Pod Details** page.



Viewing pod logs

To view logs for a pod, click **View Pod Logs** from the **Pod Details** page. By default, logs for the pod's first container are displayed, dating back to when the pod was created.



Pausing and resuming logs

Log entries are streamed in real time. New entries appear at the bottom of the log content area. Click or scroll the log content area to pause the log stream. Pausing the log stream enables you to focus on specific entries.

To resume the stream, click the **Follow Latest** button that appears after pausing.

Filtering by container

To display logs for a different container, select the container that you want from the **Container** drop-down menu.

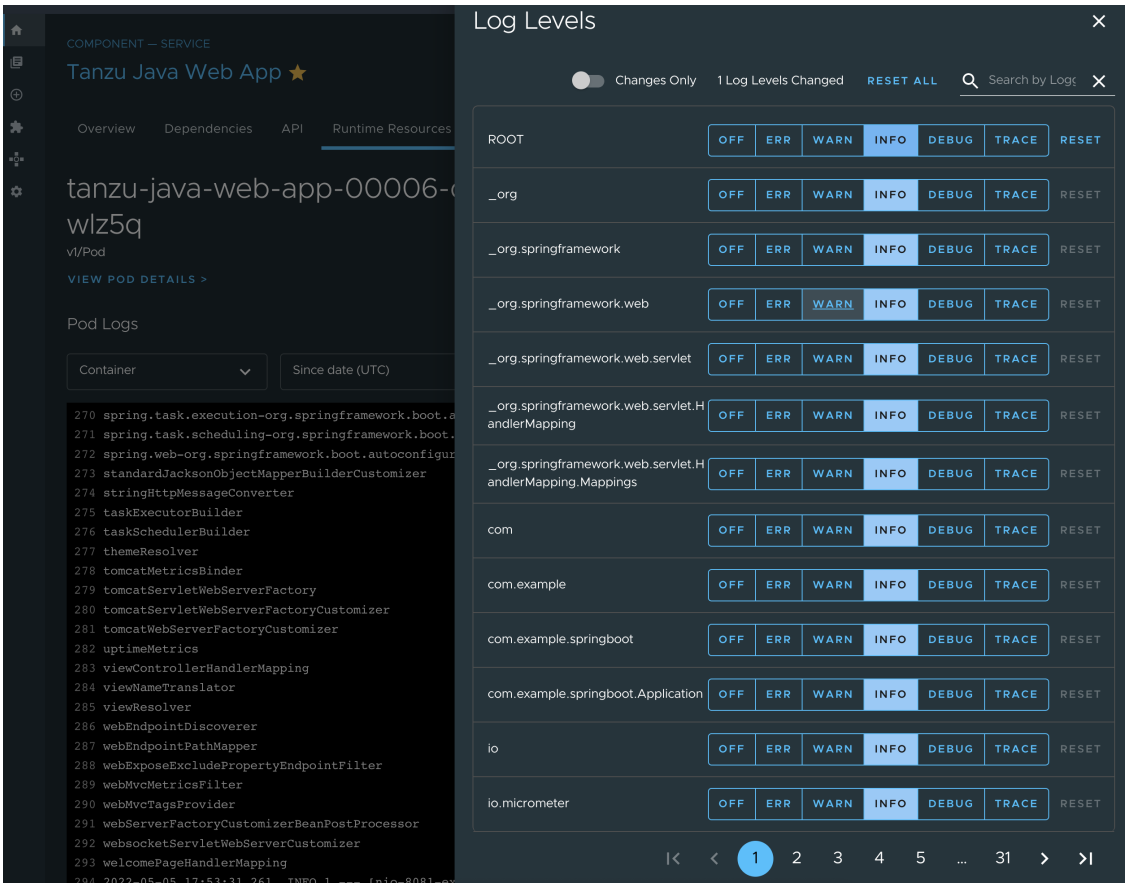
Filtering by date and time

To see logs since a specific date and time, select or type the UTC timestamp in the **Since date** field. If no logs are displayed, adjust the timestamp to an earlier time. If you do not select a timestamp, all logs produced since the pod was created are displayed.

For optimal performance, the pod logs page limits the total log entries displayed to the last 10,000, at most.

Changing log levels

If the pod is associated with an application that supports Application Live View, you can change the application's log levels by clicking the **Change Log Levels** button. You then see a panel that enables you to select levels for each logger associated with your application.



To change the levels for your application, select the desired level for each logger presented, and then click **X** in the upper-right corner of the panel, or press the Escape key, to close the panel.

Because adjusting log levels makes a real-time configuration change to your application, log-level adjustments are only reflected in log entries that your application produces after the change.

If no log entries for the expected levels appear, ensure that:

1. You adjusted the correct application loggers
2. You are viewing logs for the correct container and time frame
3. Your application is currently producing logs at the expected levels

Line wrapping

By default, log entries are not wrapped. To activate or deactivate line wrapping, click the **Wrap lines** toggle.

Downloading logs

To download current log content, click the **Download logs** button.

For optimal performance, the pod logs page limits the total log entries downloaded to the last 10,000, at most.

Connection interruptions

If the log stream connection is interrupted for any reason, such as a network error, a notification appears after the most recent log entry, and the page attempts to reconnect to the log stream. If reconnection fails, an error message displays at the top of the page, and you can click the **Refresh** button at the upper-right of the page to attempt to reconnect.

If you notice frequent disconnections at regular intervals, contact your administrator. Your administrator might need to update the back-end configuration for your installation to allow long-lived HTTP connections to log endpoints (endpoints starting with `BACKEND-HOST/api/k8s-logging/`).

Application Live View in Tanzu Application Platform GUI

This topic tells you about Application Live View in Tanzu Application Platform GUI (commonly called TAP GUI).

Overview

The Application Live View features of Tanzu Application Platform include sophisticated components to give developers and operators a view into their running workloads on Kubernetes.

Application Live View shows an individual running process, for example, a Spring Boot application deployed as a workload resulting in a JVM process running inside of a pod. This is an important concept of Application Live View. Application Live View only recognizes running processes. If there is not a running process inside of a running pod, Application Live View does not show anything.

Under the hood, Application Live View uses the concept of actuators to gather data from those running processes. It visualizes them in a semantically meaningful way and allows users to interact with the inner workings of the running processes within limited boundaries.

The actuator data serves as the source of truth. Application Live View provides a live view of the data from inside of the running processes only. It does not store any of that data for further analysis or historical views.

This easy-to-use interface provides ways to troubleshoot, learn, and maintain an overview of certain aspects of the running processes. It gives a level of control to the users to change some parameters, such as environment properties, without a restart (where the Spring Boot application, for example, supports that).

Entry point to Application Live View plug-in

The Application Live View UI plug-in is part of Tanzu Application Platform GUI. To use the Application Live View plug-in:

1. Select the relevant component under the **Organization Catalog** in Tanzu Application Platform GUI.
2. Select the desired service under the **Runtime Resources** tab.
3. Select the desired pod from the **Pods** section under the **Runtime Resources** tab.
4. You can now see all the details, do some lightweight troubleshooting, and interact with the application within certain boundaries under the **Live View** section.

Application Live View in Tanzu Application Platform GUI

This topic tells you about Application Live View in Tanzu Application Platform GUI (commonly called TAP GUI).

Overview

The Application Live View features of Tanzu Application Platform include sophisticated components to give developers and operators a view into their running workloads on Kubernetes.

Application Live View shows an individual running process, for example, a Spring Boot application deployed as a workload resulting in a JVM process running inside of a pod. This is an important concept of Application Live View. Application Live View only recognizes running processes. If there is not a running process inside of a running pod, Application Live View does not show anything.

Under the hood, Application Live View uses the concept of actuators to gather data from those running processes. It visualizes them in a semantically meaningful way and allows users to interact with the inner workings of the running processes within limited boundaries.

The actuator data serves as the source of truth. Application Live View provides a live view of the data from inside of the running processes only. It does not store any of that data for further analysis or historical views.

This easy-to-use interface provides ways to troubleshoot, learn, and maintain an overview of certain aspects of the running processes. It gives a level of control to the users to change some parameters, such as environment properties, without a restart (where the Spring Boot application, for example, supports that).

Entry point to Application Live View plug-in

The Application Live View UI plug-in is part of Tanzu Application Platform GUI. To use the Application Live View plug-in:

1. Select the relevant component under the **Organization Catalog** in Tanzu Application Platform GUI.
2. Select the desired service under the **Runtime Resources** tab.
3. Select the desired pod from the **Pods** section under the **Runtime Resources** tab.
4. You can now see all the details, do some lightweight troubleshooting, and interact with the application within certain boundaries under the **Live View** section.

Application Live View for Spring Boot applications in Tanzu Application Platform GUI

This topic tells you about the Application Live View pages for Spring Boot Applications in Tanzu Application Platform GUI (commonly called TAP GUI).

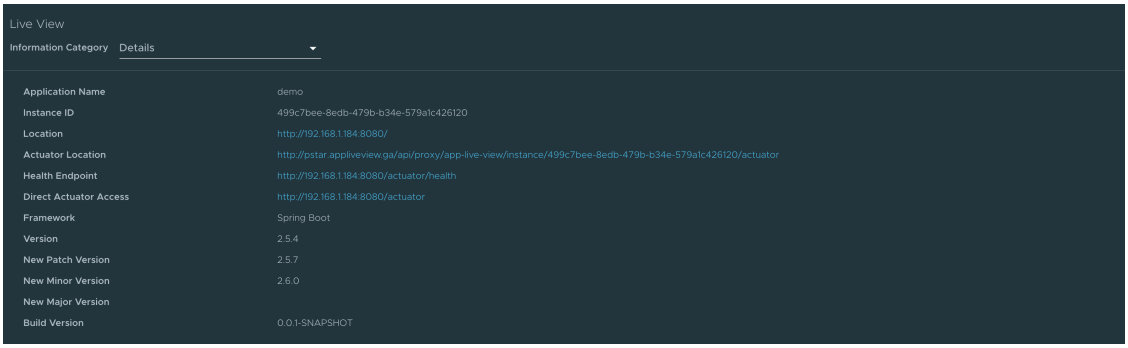
Details page

This is the default page loaded in the **Live View** section. This page gives a tabular overview containing the following information:

- application name
- instance ID
- location
- actuator location
- health endpoint
- direct actuator access
- framework
- version
- new patch version
- new major version

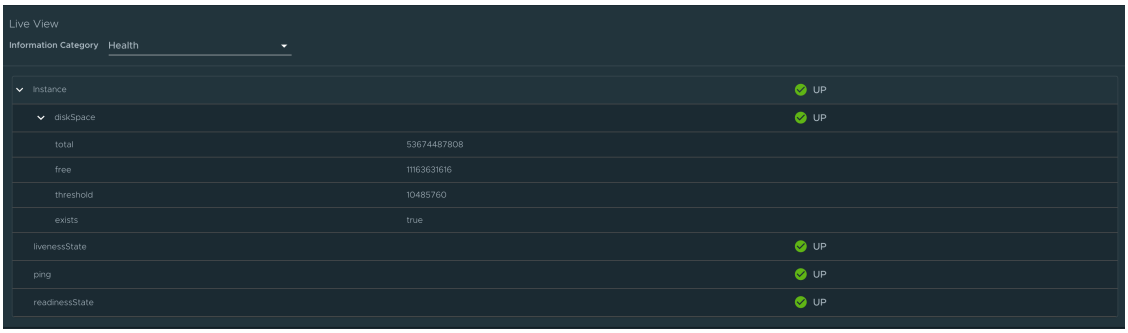
- build version

You can navigate between **Information Categories** by selecting from the drop-down menu on the top right corner of the page.



Health page

To go to the health page, select the **Health** option from the **Information Category** drop-down menu. The health page provides detailed information about the health of the application. It lists all the components that make up the health of the application such as readiness, liveness, and disk space. It displays the status and details associated with each component.



Environment page

To go to the **Environment** page, select the **Environment** option from the **Information Category** drop-down menu. The Environment page contains details of the applications' environment. It contains properties including, but not limited to, system properties, environment variables, and configuration properties (such as application.properties) in a Spring Boot application.

The page includes the following capabilities for `viewing` configured environment properties:

- The UI has a search feature that enables you to search for a property or values.
- Each property has a search icon at the right corner which helps you quickly see all the occurrences of a specific property key without manually typing in the search box. Clicking the search button locates the property name.
- The **Refresh Scope** button on the top right corner of the page probes the application to refresh all the environment properties.

The page also includes the following capabilities for `editing` configured environment properties:

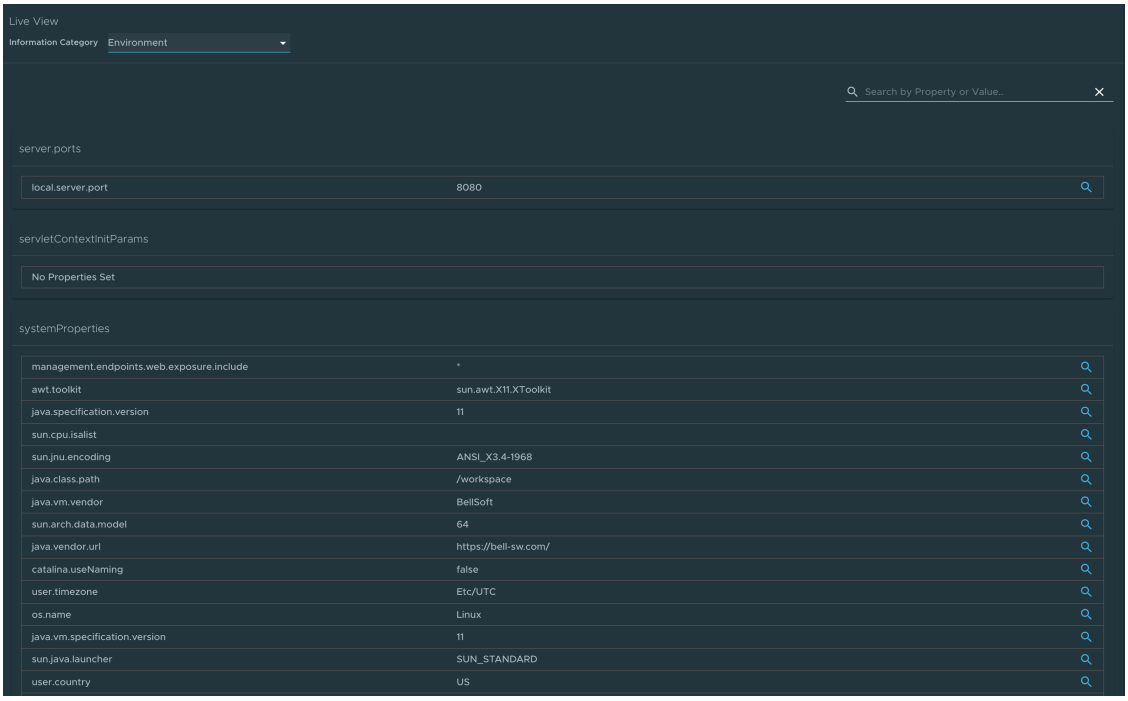
- The UI allows you to edit environment properties and see the live changes in the application. These edits are temporary and go away if the underlying pod is restarted.
- For each configured environment property, you can edit its value by clicking on the **Override** button in the same row. After the value is saved, you can view the message that the property was overridden from the initial value. The updated property is visible in the

Applied Overrides section at the top of the page. The **Reset** button in the same row resets the environment property to the initial state.

- You can edit or remove the overridden environment variables in the **Applied Overrides** section.
- The **Applied Overrides** section also enables you to add new environment properties to the application.

Note

`management.endpoint.env.post.enabled=true` must be set in the application config properties of the application and a corresponding, editable environment must be present in the application.



Log Levels page

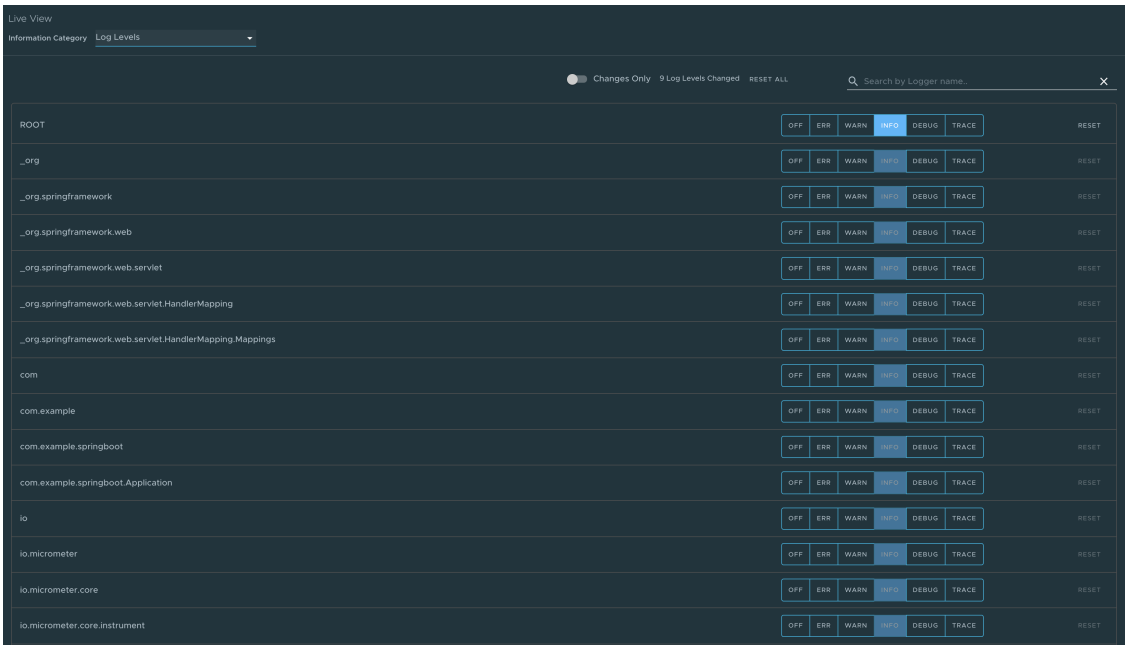
To go to the **Log Levels** page, select the **Log Levels** option from the **Information Category** drop-down menu. The log levels page provides access to the application's loggers and the configuration of their levels.

You can configure the log levels such as INFO, DEBUG, and TRACE in real time from the UI. You can search for a package and edit its respective log level. You can configure the log levels at a specific class and package. They can deactivate all the log levels by modifying the log level of root logger to OFF.

The toggle **Changes Only** displays the changed log levels. Use the search feature to search by logger name. The **Reset** resets the log levels to the original state. The **Reset All** on top right corner of the page resets all the loggers to default state.

Note

Use the UI to change the log levels and see the live changes on the application. These changes are temporary and go away if the underlying pod is restarted.

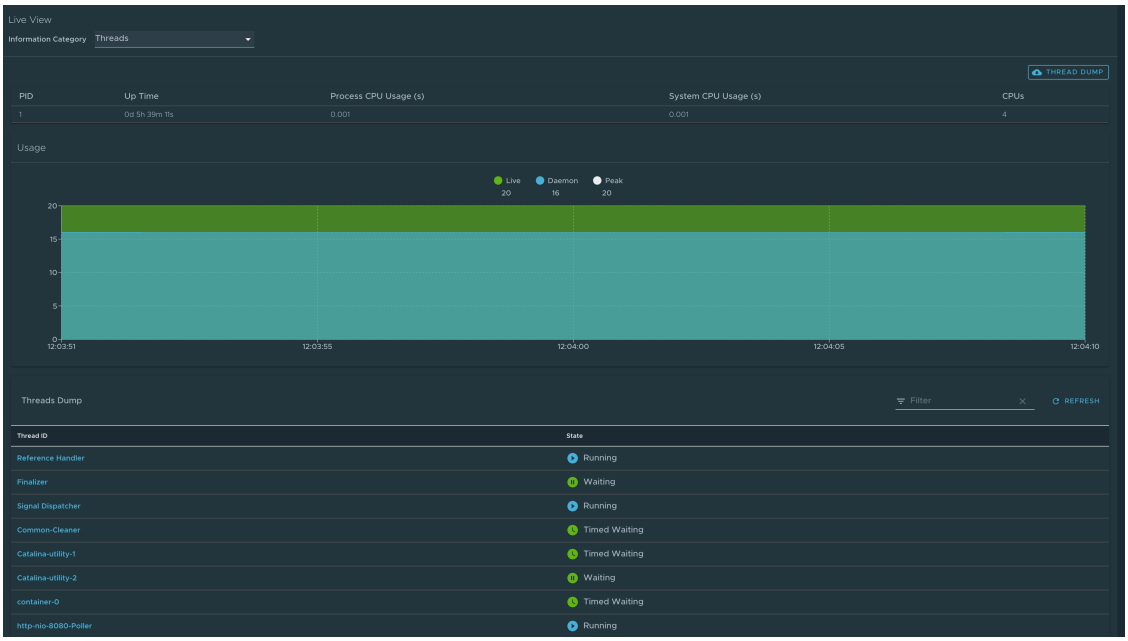


Threads page

To go to the **Threads** page, select the **Threads** option from the **Information Category** drop-down menu.

This page displays all details related to Java Virtual Machine (JVM) threads and running processes of the application. This tracks live threads and daemon threads real-time. It is a snapshot of different thread states. Navigating to a thread state displays all the information about a particular thread and its stack trace.

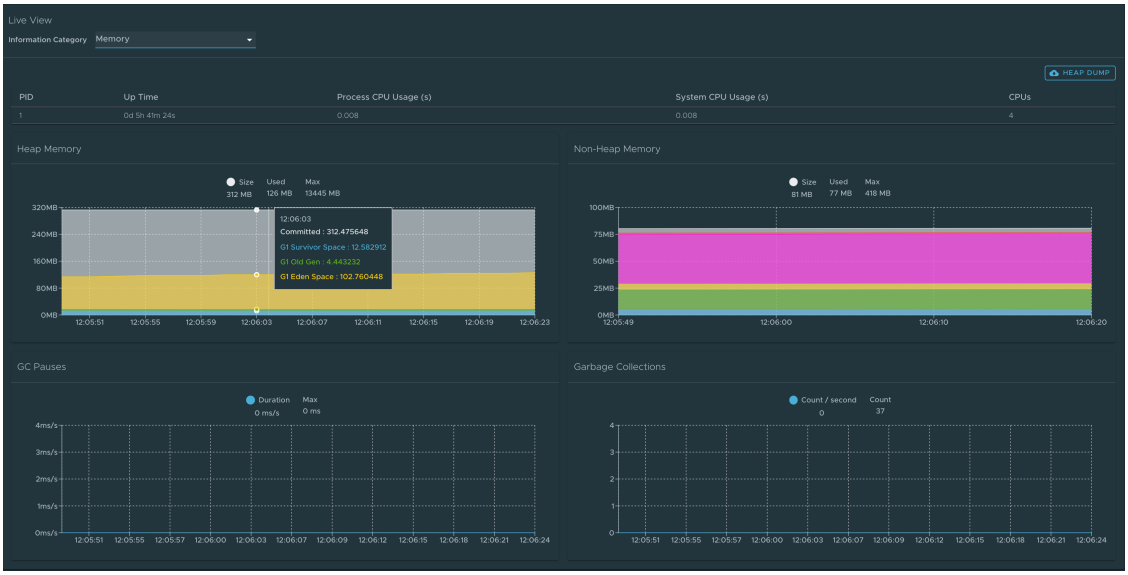
Use the search feature to search for threads by thread ID or state. The refresh icon refreshes to the latest state of the threads. You can view more thread details by clicking on the Thread ID. The page also has a feature to download thread dump data for analysis purposes.



Memory page

To go to the **Memory** page, select the **Memory** option from the **Information Category** drop-down menu.

- The memory page highlights the memory use inside of the JVM. It displays a graphical representation of the different memory regions within heap and non-heap memory. This visualizes data from inside of the JVM (in case of Spring Boot apps running on a JVM) and therefore provides memory insights into the application in contrast to “outside” information about the Kubernetes pod level.
- The real-time graphs displays a stacked overview of the different spaces in memory with the total memory used and total memory size. The page contains graphs to display the GC pauses and GC events.
- The **Heap Dump** at the top-right corner enables you to download heap dump data.



Note

This graphical visualization happens in real time and shows real-time data only. As mentioned at the top, the Application Live View features do not store any information. That means the graphs visualize the data over time only for as long as you stay on that page.

Request Mappings page

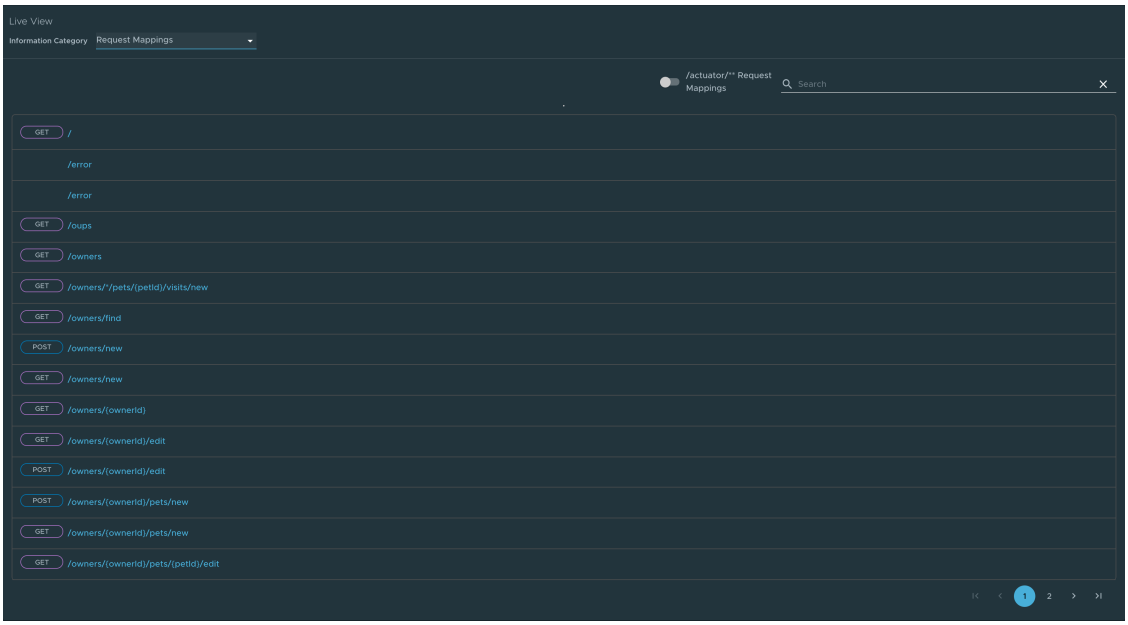
To go to the Request Mappings page, select the **Request Mappings** option from the **Information Category** drop-down menu.

This page provides information about the application’s request mappings. For each mapping, it displays the request handler method. You can view more details of the request mapping, such as the header metadata of the application.

When you click on the request mapping, a side panel appears. This panel contains information about the mapping-media types `Produces` and `Consumes`. The panel also displays the `Handler` class for the request. Use the search feature to search for the request mapping or the method. The toggle **/actuator/** Request Mappings** displays the actuator related mappings of the application.

Note

When application actuator endpoint is exposed on `management.server.port`, the application does not return any actuator request mappings data in the context. The application displays a message when the actuator toggle is enabled.



HTTP Requests page

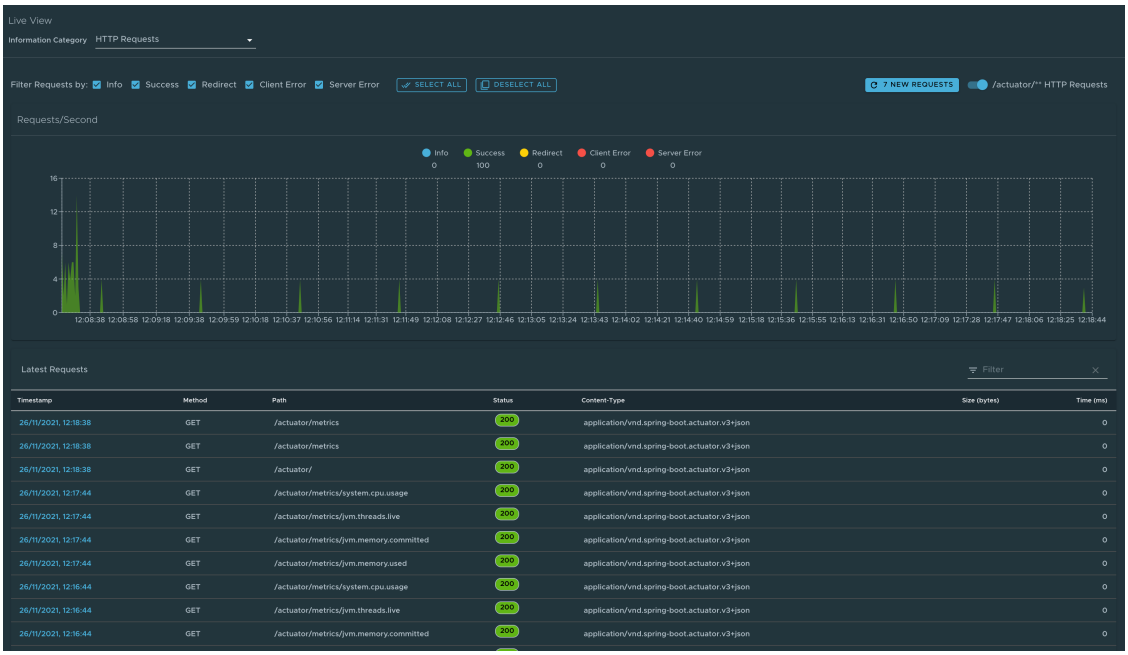
To go to the HTTP Requests page, click **HTTP Requests** from the **Information Category** drop-down menu. The HTTP Requests page provides information about HTTP request-response exchanges to the application.

The graph visualizes the requests per second indicating the response status of all the requests. You can filter the response statuses, which include info, success, redirects, client-errors, and server-errors. The trace data is captured in detail in a tabular format with metrics such as timestamp, method, path, status, content-type, length, time.

The search feature on the table filters the traces based on the search field value. You can view more details of the request, such as method, headers, and response of the application by clicking on the timestamp. The refresh icon above the graph loads the latest traces of the application. The toggle **/actuator/**** on the top right corner of the page displays the actuator related traces of the application.

Note

When application actuator endpoint is exposed on management.server.port, no actuator HTTP Traces data is returned for the application. In this case, a message is displayed when the actuator toggle is enabled.



Caches page

To go to the **Caches** page, select the **Caches** option from the **Information Category** drop-down menu.

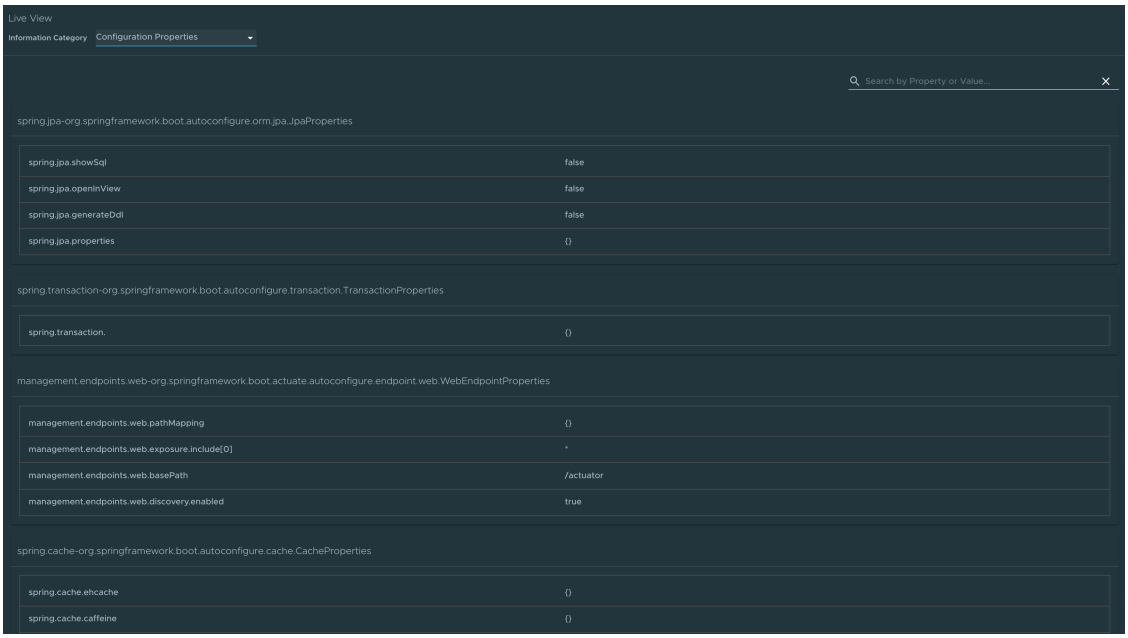
The Caches page provides access to the application's caches. It gives the details of the cache managers associated with the application including the fully qualified name of the native cache.

Use the search feature in the Caches Page to search for a specific cache/cache manager. You can clear individual caches by clicking **Evict**. You can clear all the caches completely by clicking **Evict All**. If there are no cache managers for the application, the message `No cache managers available for the application` is displayed.

Configuration Properties page

To go to the **Configuration Properties** page, select the **Configuration Properties** option from the **Information Category** drop-down menu.

The configuration properties page provides information about the configuration properties of the application. In case of Spring Boot, it displays application's `@ConfigurationProperties` beans. It gives a snapshot of all the beans and their associated configuration properties. Use the search feature to search for a property's key/value or the bean name.

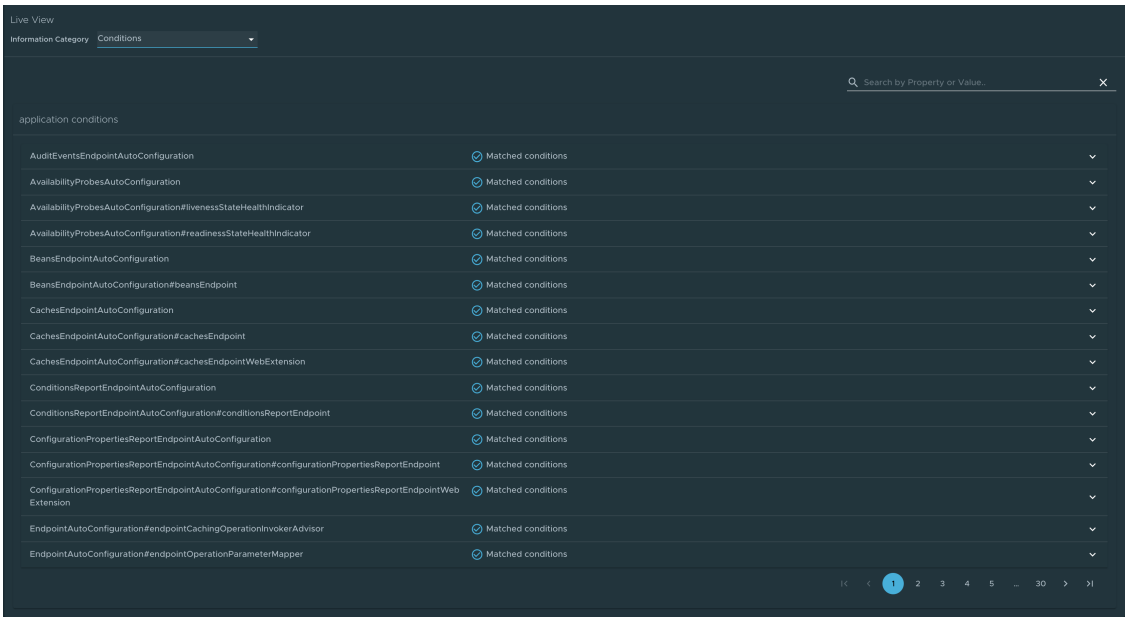


Conditions page

To go to the **Conditions** page, select the **Conditions** option from the **Information Category** drop-down menu. The conditions evaluation report provides information about the evaluation of conditions on configuration and auto-configuration classes.

In the case of Spring Boot, this gives you a view of all the beans configured in the application. When you click on the bean name, the conditions and the reason for the conditional match is displayed.

In the case of non-configured beans, it shows both the matched and unmatched conditions of the bean if any. In addition to this, it also displays names of unconditional auto configuration classes if any. You can use the search feature to filter out the beans and the conditions.

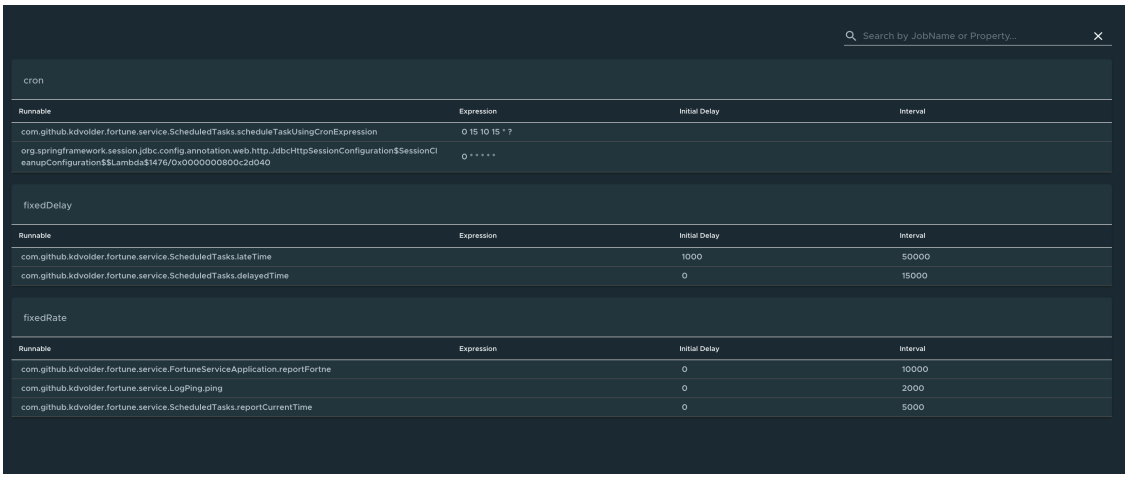


Scheduled Tasks page

To go to the **Scheduled Tasks** page, select the **Scheduled Tasks** option from the **Information Category** drop-down menu.

The scheduled tasks page provides information about the application's scheduled tasks. It includes cron tasks, fixed delay tasks and fixed rate tasks, custom tasks and the properties associated with them.

You can search for a particular property or a task in the search bar to retrieve the task or property details.



Beans page

To go to the **Beans** page, select the **Beans** option from the **Information Category** drop-down menu. The beans page provides information about a list of all application beans and its dependencies. It displays the information about the bean type, dependencies, and its resource. You can search by the bean name or its corresponding fields.

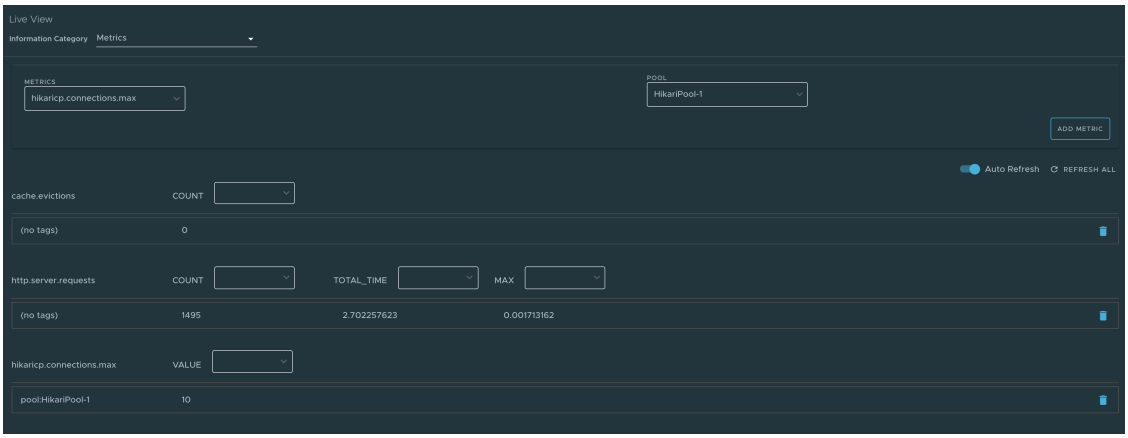
Metrics page

To go to the **Metrics** page, select the **Metrics** option from the **Information Category** drop-down menu.

The metrics page provides access to application metrics information. You can choose from the list of various metrics available for the application, such as `jvm.memory.used`, `jvm.memory.max`, `http.server.request`, and so on.

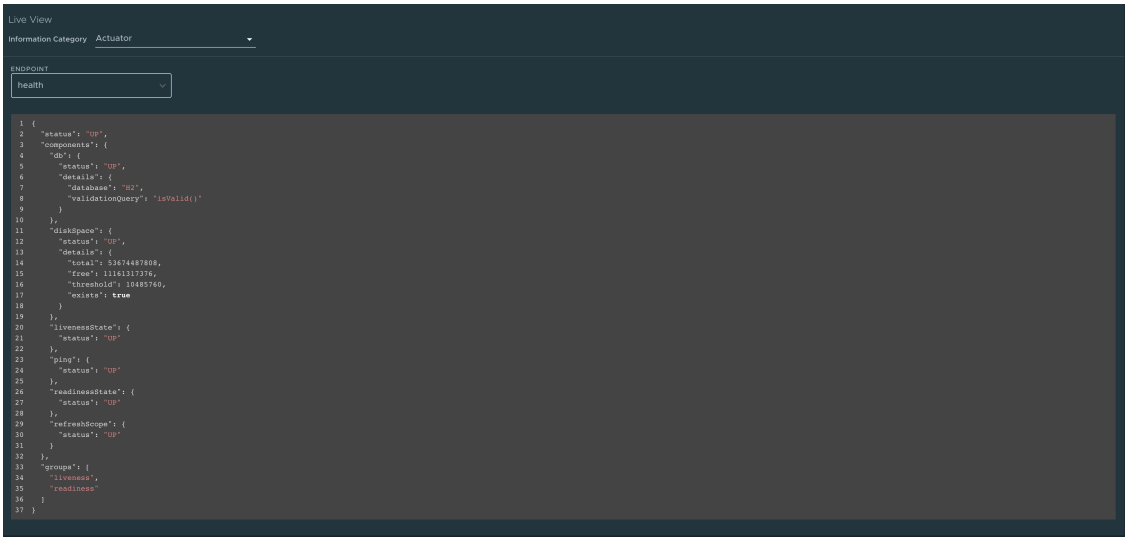
After the metric is chosen, you can view the associated tags. You can choose the value of each tag based on filtering criteria. Clicking **Add Metric** adds the metric to the page which is refreshed every 5 seconds by default.

You can pause the auto refresh feature by deactivating the **Auto Refresh** toggle. You can refresh the metrics manually by clicking **Refresh All**. The format of the metric value can be changed according to your needs. They can delete a particular metric by clicking the minus symbol in the same row.



Actuator page

To go to the **Actuator** page, select the **Actuator** option from the **Information Category** drop-down menu. The actuator page provides a tree view of the actuator data. You can choose from a list of actuator endpoints and parse through the raw actuator data.



Troubleshooting

You might run into cases where a workload running on your cluster does not appear in the Application Live View overview, the detail pages do not load any information while running, or similar issues. If you encounter issues, see Troubleshooting in the Application Live View documentation.

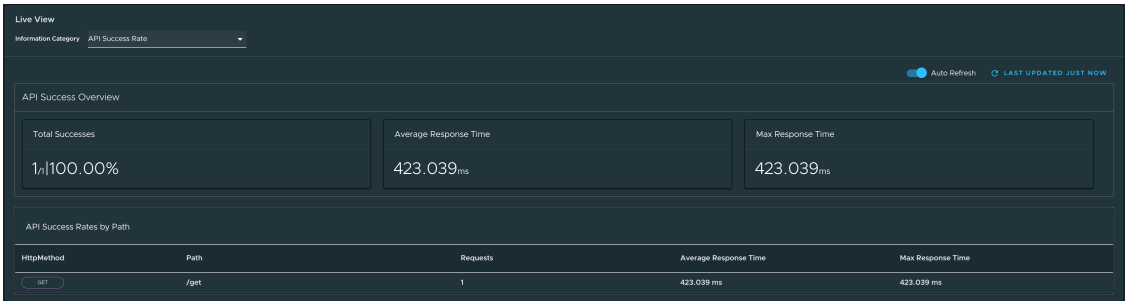
Application Live View for Spring Cloud Gateway applications in Tanzu Application Platform GUI

This topic tells you about the Application Live View pages for Spring Cloud Gateway applications in Tanzu Application Platform GUI (commonly called TAP GUI).

API Success Rate page

To access to the API Success Rate page, select the **API Success Rate** option from the **Information Category** drop-down menu.

The API success rate page displays the total successes, average response time, and maximum response time for the gateway routes. It also displays the details of each successful route path.



API Overview page

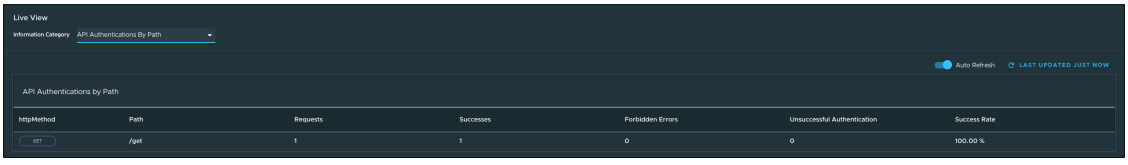
To access the API Overview page, select the **API Overview** option from the **Information Category** drop-down menu.

The API Overview page provides route count, number of successes, errors, and the rate-limited requests. It also provides an **auto refresh** feature to get the updated results. These metrics are depicted in a line graph.

API Authentications By Path page

To access the API Authentications By Path page, select the **API Authentications By Path** option from the **Information Category** drop-down menu.

The API Authentications By Path page displays the total requests, number of successes, and forbidden and unsuccessful authentications grouped by the HTTP method and gateway route path. The page also displays the success rate for each route.



Note

In addition to the preceding three pages, the Spring Boot actuator pages are also displayed.

Troubleshooting

You might run into cases where a workload running on your cluster does not appear in the Application Live View overview, or the detail pages do not load any information while running, or other similar issues. For more information about such issues, see Troubleshooting in the Application Live View documentation.

Application Live View for Steeltoe applications in Tanzu Application Platform GUI

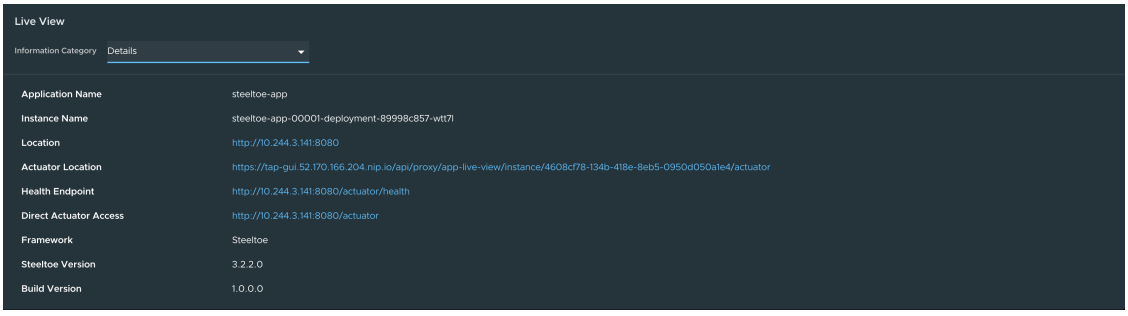
This topic tells you about the Application Live View pages for Steeltoe applications in Tanzu Application Platform GUI (commonly called TAP GUI).

Details page

This is the default page loaded in the **Live View** section. This page gives a tabular overview containing the following information:

- Application name
- Instance ID
- Location
- Actuator location
- Health endpoint
- Direct actuator access
- Framework
- Version
- New patch version
- New major version
- Build version

You can navigate between **Information Categories** by selecting from the drop-down menu on the top right corner of the page.



Health page

To access the health page, select the **Health** option from the **Information Category** drop-down menu.

The health page provides detailed information about the health of the application. It lists all the components that make up the health of the application, such as readiness, liveness, and disk space. It displays the status and details associated with each component.

Environment page

To access the **Environment** page, select the **Environment** option from the **Information Category** drop-down menu.

The Environment page contains details of the applications' environment. It contains properties including, but not limited to, system properties, environment variables, and configuration properties (such as `appsettings.json`) in a Steeltoe application.

The page includes the following capabilities for `viewing` configured environment properties:

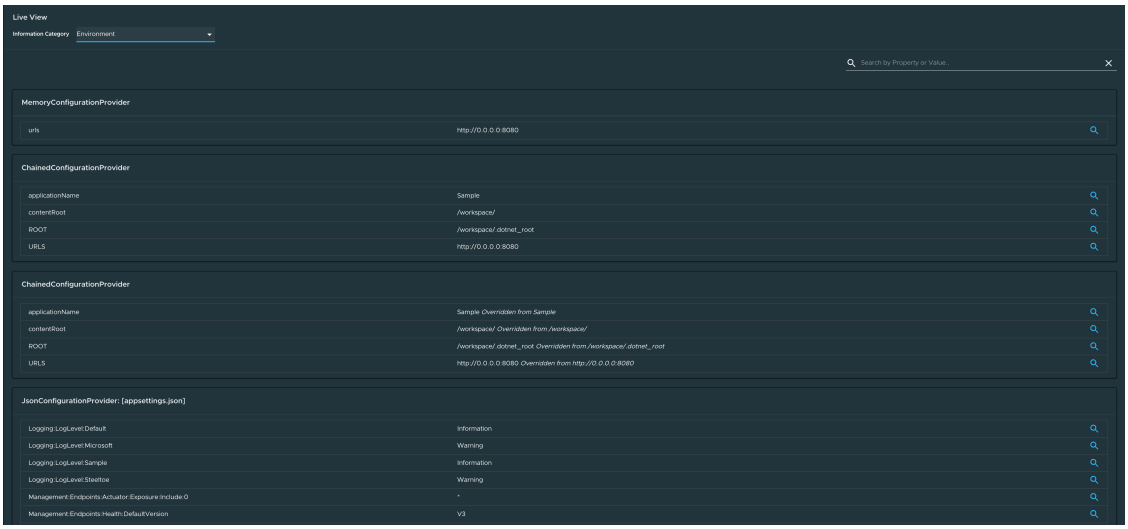
- The UI has a search feature that enables you to search for a property or values.
- Each property has a search icon at the right corner which helps you quickly see all the occurrences of a specific property key without manually typing in the search box. Clicking the search button locates the property name.
- The **Refresh Scope** button on the top right corner of the page probes the application to refresh all the environment properties.

The page also includes the following capabilities for `editing` configured environment properties:

- The UI allows you to edit environment properties and see the live changes in the application. These edits are temporary and go away if the underlying pod is restarted.
- For each of the configured environment properties, you can edit its value by clicking on the **Override** button in the same row. After the value is saved, you can view the message that the property was overridden from the initial value. Also, the updated property is visible in the **Applied Overrides** section at the top of the page. The **Reset** button in the same row resets the environment property to the initial state.
- You can also edit or remove the overridden environment variables in the **Applied Overrides** section.
- The **Applied Overrides** section also enables you to add new environment properties to the application.

Note

The `management.endpoint.env.post.enabled=true` must be set in the application config properties of the application, and a corresponding editable environment must be present in the application.



Log Levels page

To go to the **Log Levels** page, select the **Log Levels** option from the **Information Category** drop-down menu. The **Log Levels** page provides access to the application's loggers and the configuration of the levels.

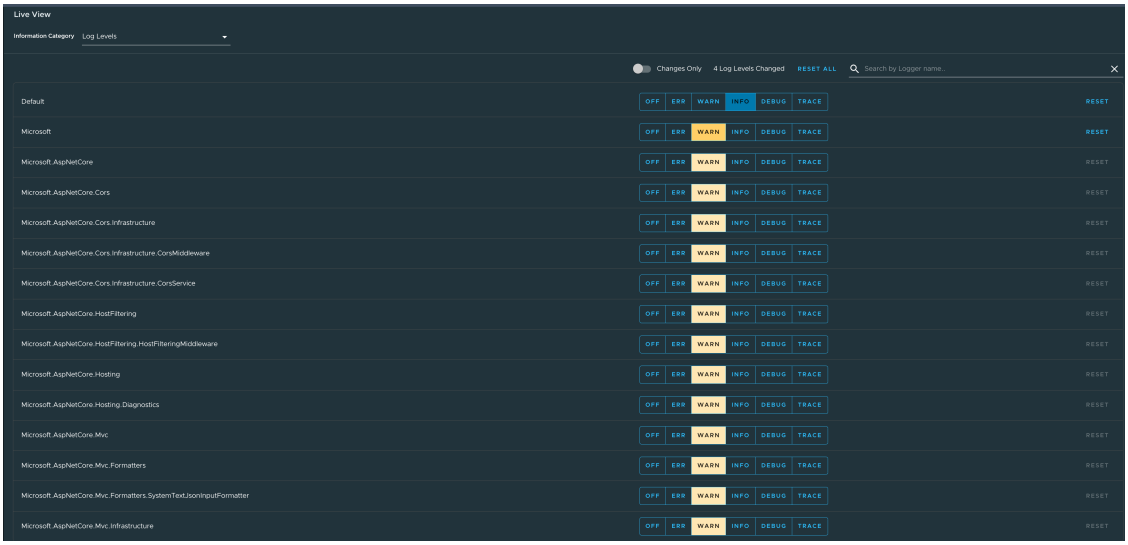
You can:

- Configure log levels, such as **INFO**, **DEBUG**, and **TRACE**, in real time from the UI
- Search for a package and edit its respective log level
- Configure the log levels at a specific class and package
- Deactivate all the log levels by changing the log level of root logger to **OFF**

Use the **Changes Only** toggle to display the changed log levels. Use the search feature to search by logger name. Click **Reset All** to reset all the loggers to the default state.

Note

The UI allows you to change the log levels and see the live changes on the application. These changes are temporary and go away if the underlying pod is restarted.

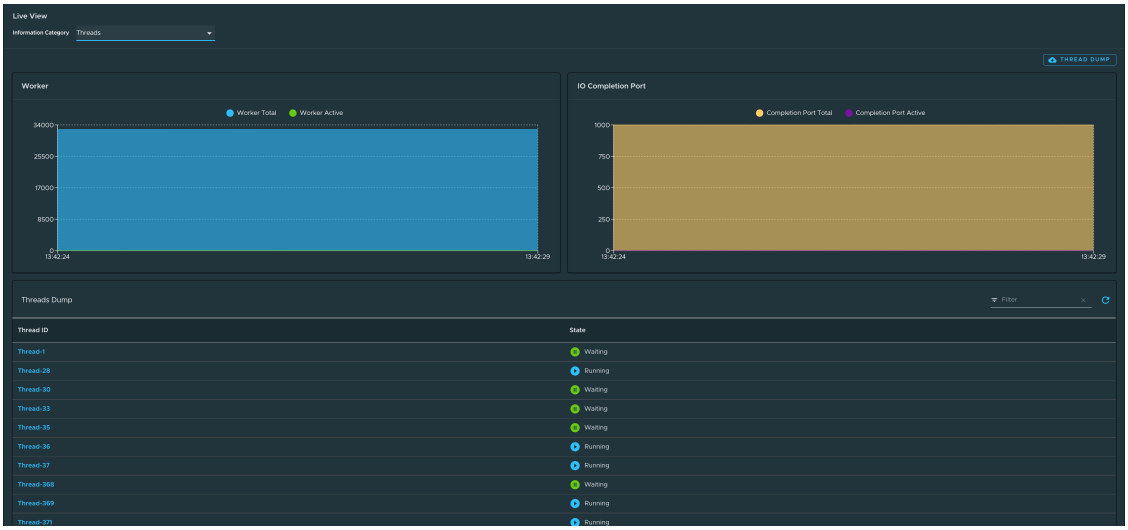


Threads page

To access the **Threads** page, select the **Threads** option from the **Information Category** drop-down menu.

This page displays all details related to CLR threads and running processes of the application. This tracks worker threads and completion port threads in real time. Navigating to a thread state displays all the information about a particular thread and its stack trace.

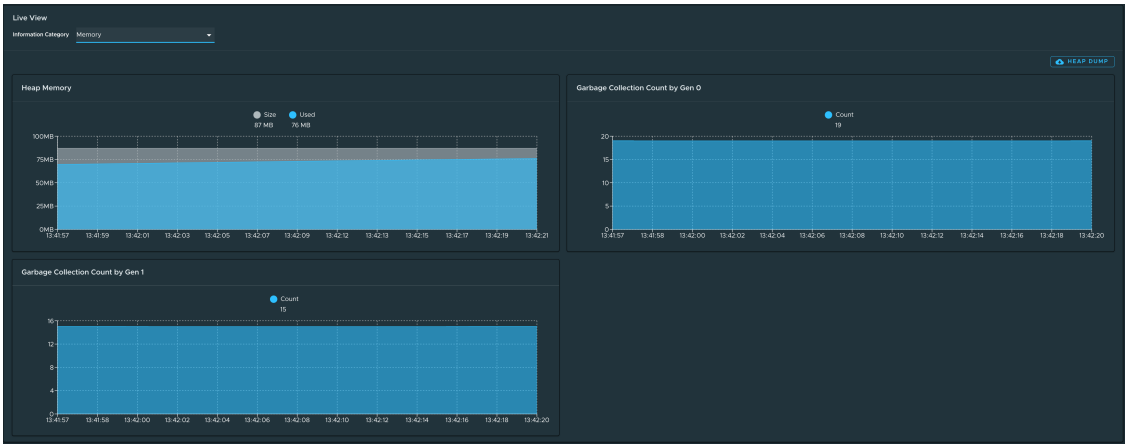
- The refresh icon refreshes to the latest state of the threads.
- To view more thread details, click the thread ID.
- The page has a feature to download the thread dump for analysis.
- The page has a feature to view the CPU stats for a Steeltoe application.



Memory page

To access the **Memory** page, select the **Memory** option from the **Information Category** drop-down menu.

This page displays all details related to used and committed memory of the application. This also displays the garbage collection count by generation (gen0/gen1). The page also has a feature to download the heap dump for analysis. The page also has a feature to view the CPU stats for a Steeltoe application

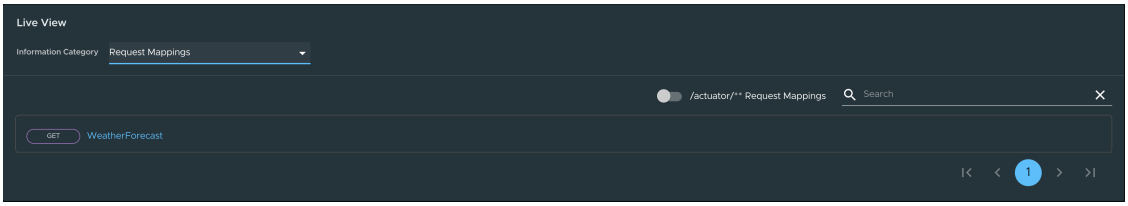


Request Mappings page

To access the **Request Mappings** page, select the **Request Mappings** option from the **Information Category** drop-down menu.

This page provides information about the application's request mappings. For each mapping, the page displays the request handler method. You can view more details of the request mapping, such as the header metadata of the application.

When you click on the request mapping, a side panel appears. This panel contains information about the mapping-media types `Produces` and `Consumes`. The panel also displays the `Handler` class for the request. The search feature enables you to search for the request mapping or the method. The toggle **/actuator/** Request Mappings** displays the actuator-related mappings of the application.



HTTP Requests page

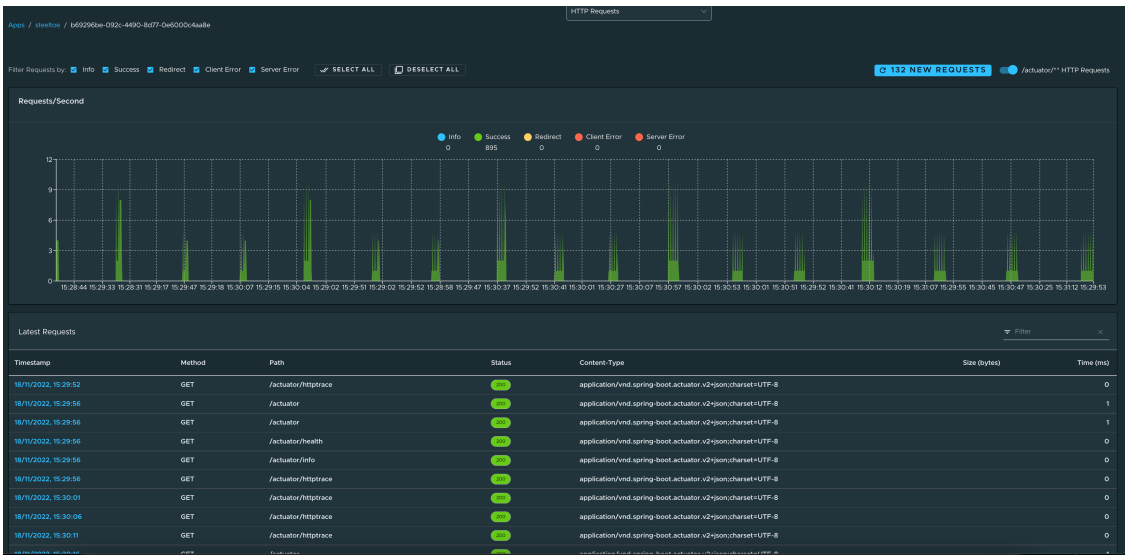
To access the **HTTP Requests** page, select the **HTTP Requests** option from the **Information Category** drop-down menu.

The **HTTP Requests** page provides information about HTTP request-response exchanges to the application.

The graph visualizes the requests per second, which indicates the response status of all the requests. You can filter by the response statuses, which include **info**, **success**, **redirects**, **client-errors**, and **server-errors**. The trace data is captured in detail in a tabular format with metrics, such as **timestamp**, **method**, **path**, **status**, **content-type**, **length**, and **time**.

The search feature on the table filters the traces based on the search text box value. By clicking on the timestamp, you can view more details of the request, such as method, headers, and the response of the application.

The refresh icon above the graph loads the latest traces of the application. The toggle **/actuator/**** on the top-right corner of the page displays the actuator-related traces of the application.



Metrics page

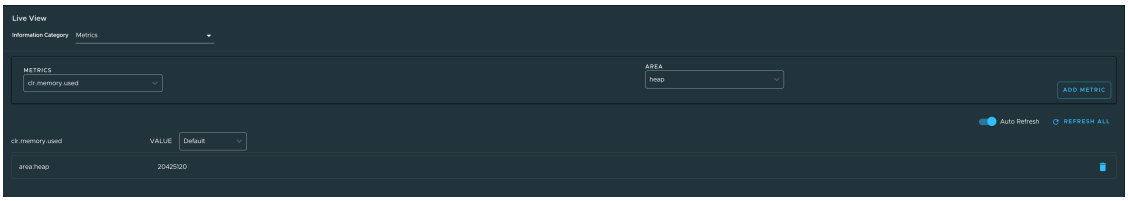
To access the **Metrics** page, select the **Metrics** option from the **Information Category** drop-down menu.

The metrics page provides access to application metrics information. You can choose from the list of various metrics available for the application, such as `clr.memory.used`, `System.Runtime.gc-committed`, `clr.threadpool.active`, and so on.

After you choose the metric, you can view the associated tags. You can choose the value of each of the tags based on filtering criteria. Click **Add Metric** to add the metric to the page. The page is refreshed every 5 seconds by default.

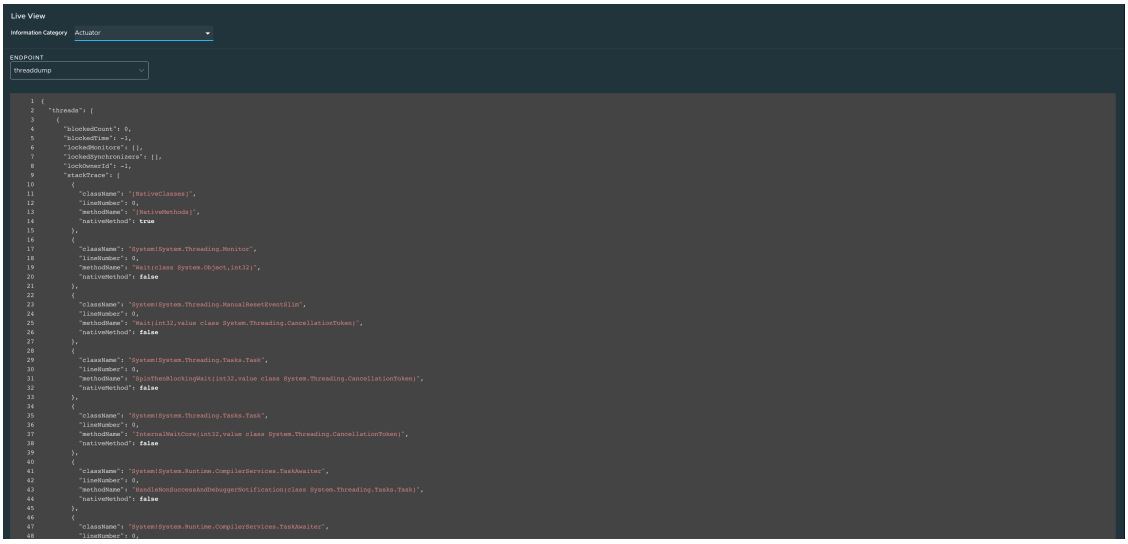
The UI on the **Metrics** page includes features that enable you to:

- Pause the auto refresh feature by deactivating the **Auto Refresh** toggle.
- Refresh the metrics manually by clicking **Refresh All**.
- Change the format of the metric value according to your needs.
- Delete a particular metric by clicking the minus-sign button in the relevant row.



Actuator page

To access the **Actuator** page, select the **Actuator** option from the **Information Category** drop-down menu. The actuator page provides a tree view of the actuator data. You can choose from a list of actuator endpoints and parse through the raw actuator data.



Troubleshooting

You might run into cases where a workload running on your cluster does not appear in the Application Live View overview, or the Details pages do not load any information while running, or other similar issues. For help with troubleshooting common issues, see Troubleshooting.

Application Accelerator in Tanzu Application Platform GUI

This topic tells you how to use Application Accelerator in Tanzu Application Platform GUI (commonly called TAP GUI).

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing and deploying your applications in a discoverable and repeatable way.

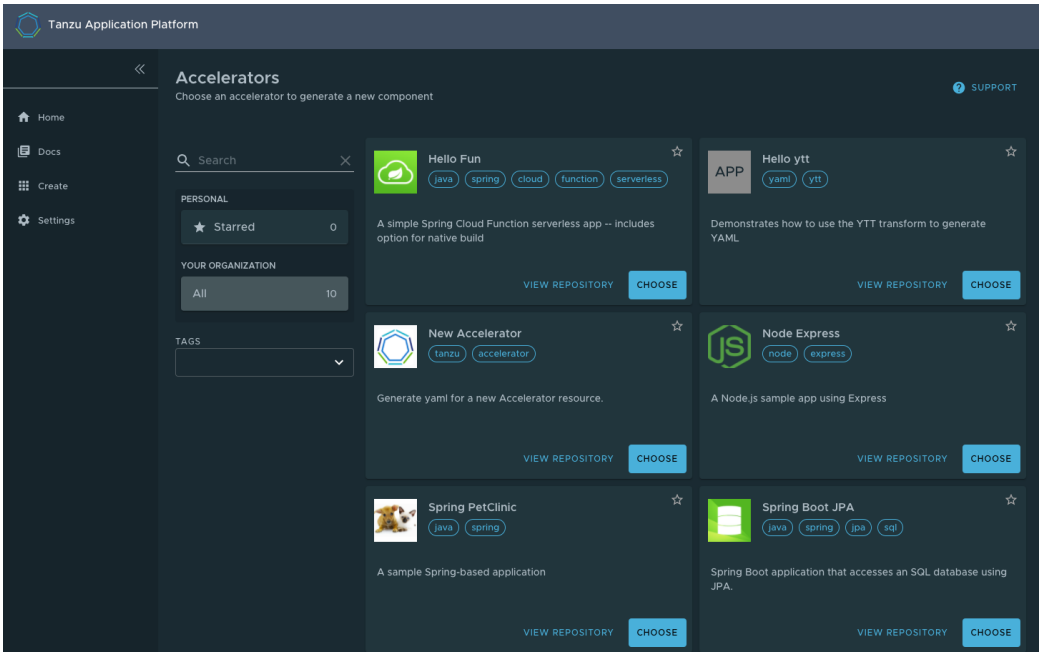
Enterprise architects author and publish accelerator projects that provide developers and operators with ready-made, enterprise-conforming code and configurations. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

Access Application Accelerator

To open the Application Accelerator UI plug-in and select an accelerator:

1. Within Tanzu Application Platform, click **Create** in the left navigation pane to open the **Accelerators** page.



   Here you can view accelerators already registered with the system. Developers can add new accelerators by registering them with Kubernetes.

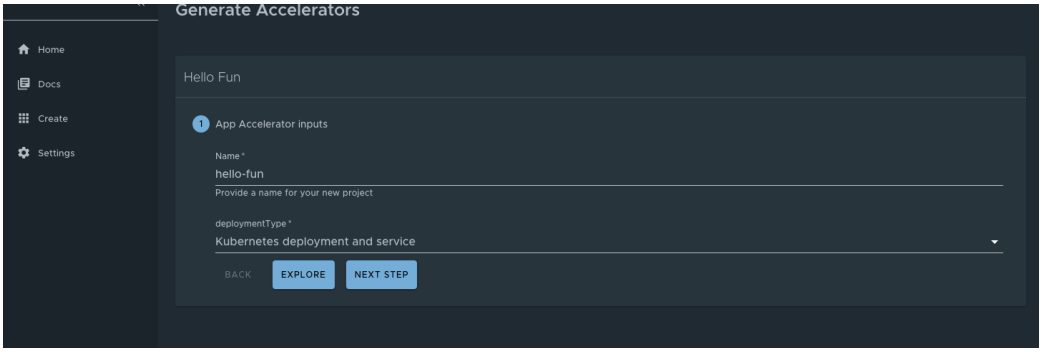
2. Every accelerator has a title and short description. Click **VIEW REPOSITORY** to view an accelerator definition. This opens the accelerator's Git repository in a new browser tab.
3. Search and filter based on text and tags associated with the accelerators to find the accelerator representing the project you want to create.
4. Click **CHOOSE** for the accelerator you want. This opens the **Generate Accelerators** page.

Configure project generation

To configure how projects are generated:

1. On the **Generate Accelerators** page, add any configuration values needed to generate the project. The application architect defined these values in `accelerator.yaml` in the

accelerator definition. Filling some text boxes can cause other text boxes to appear. Fill them all in.

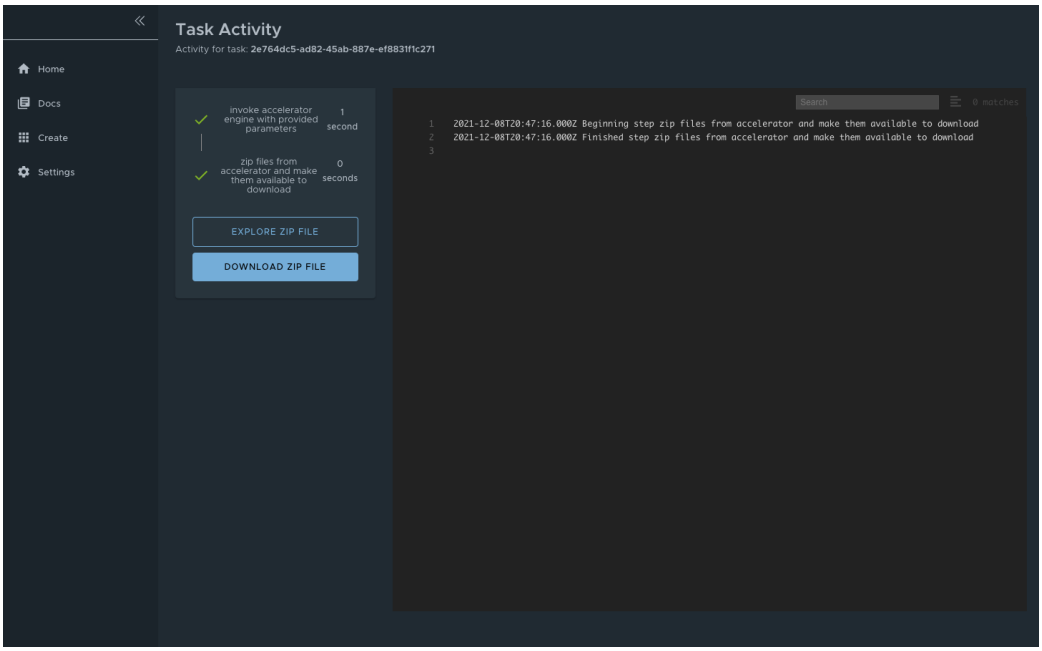


2. Click **EXPLORE** to open the **Explore Project** page and view the project before it is generated.
3. After configuring your project, click **NEXT STEP** to see the project summary page.
4. Review the values you specified for the configurable options.
5. Click **BACK** to make more changes, if necessary. Otherwise, proceed to create the project.

Create the project

To create the project:

1. Click **Create** to start generating your project. See the progress on the **Task Activity** page. A detailed log is displayed on the right.



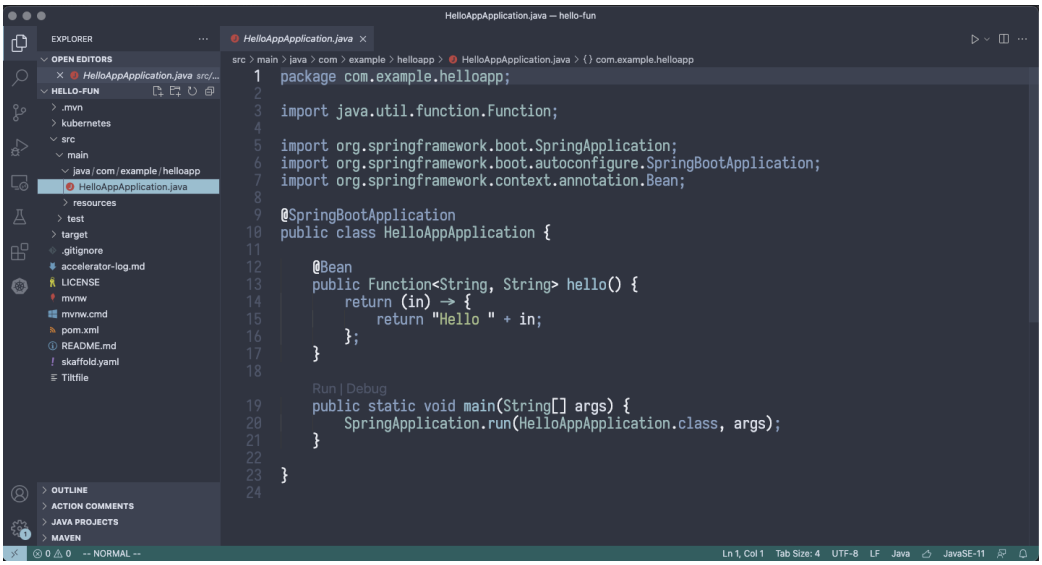
2. After the project is generated, click **EXPLORE ZIP FILE** to open the **Explore Project** page to verify configuration.
3. Click **DOWNLOAD ZIP FILE** to download the project in a ZIP file.

Develop your code

To develop your code:

1. Expand the ZIP file.

2. Open the project in your integrated development environment (IDE).



Next steps

To learn more about Application Accelerator for VMware Tanzu, see the Application Accelerator documentation.

Application Accelerator in Tanzu Application Platform GUI

This topic tells you how to use Application Accelerator in Tanzu Application Platform GUI (commonly called TAP GUI).

Overview

Application Accelerator for VMware Tanzu helps you bootstrap developing and deploying your applications in a discoverable and repeatable way.

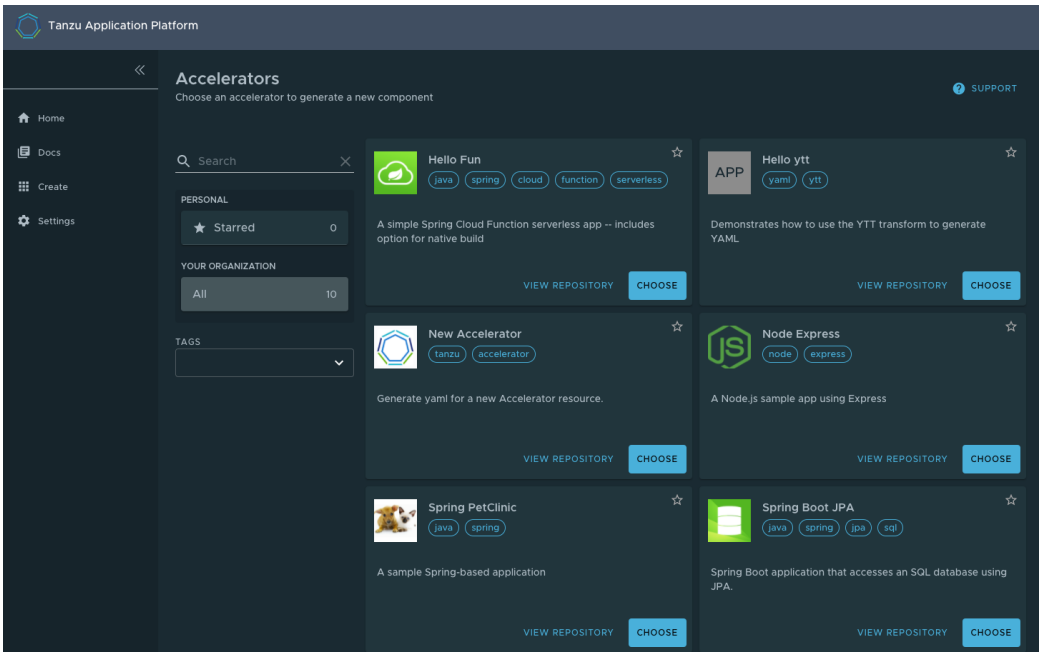
Enterprise architects author and publish accelerator projects that provide developers and operators with ready-made, enterprise-conforming code and configurations. You can then use Application Accelerator to create new projects based on those accelerator projects.

The Application Accelerator UI enables you to discover available accelerators, configure them, and generate new projects to download.

Access Application Accelerator

To open the Application Accelerator UI plug-in and select an accelerator:

1. Within Tanzu Application Platform, click **Create** in the left navigation pane to open the **Accelerators** page.



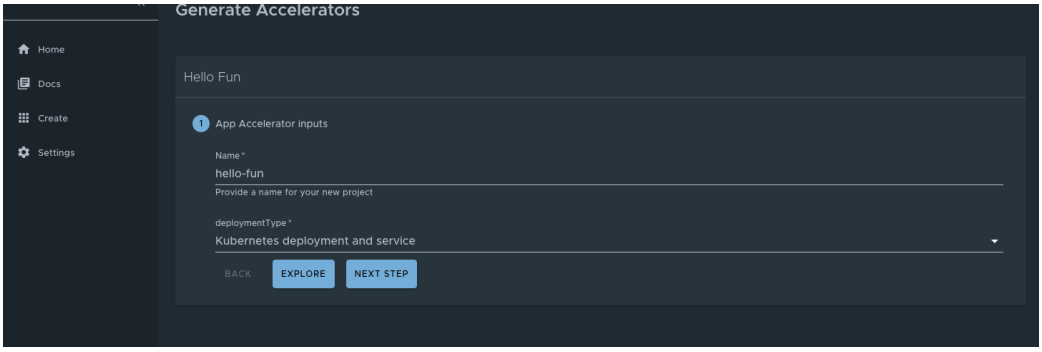
Here you can view accelerators already registered with the system. Developers can add new accelerators by registering them with Kubernetes.

2. Every accelerator has a title and short description. Click **VIEW REPOSITORY** to view an accelerator definition. This opens the accelerator's Git repository in a new browser tab.
3. Search and filter based on text and tags associated with the accelerators to find the accelerator representing the project you want to create.
4. Click **CHOOSE** for the accelerator you want. This opens the **Generate Accelerators** page.

Configure project generation

To configure how projects are generated:

1. On the **Generate Accelerators** page, add any configuration values needed to generate the project. The application architect defined these values in `accelerator.yaml` in the accelerator definition. Filling some text boxes can cause other text boxes to appear. Fill them all in.

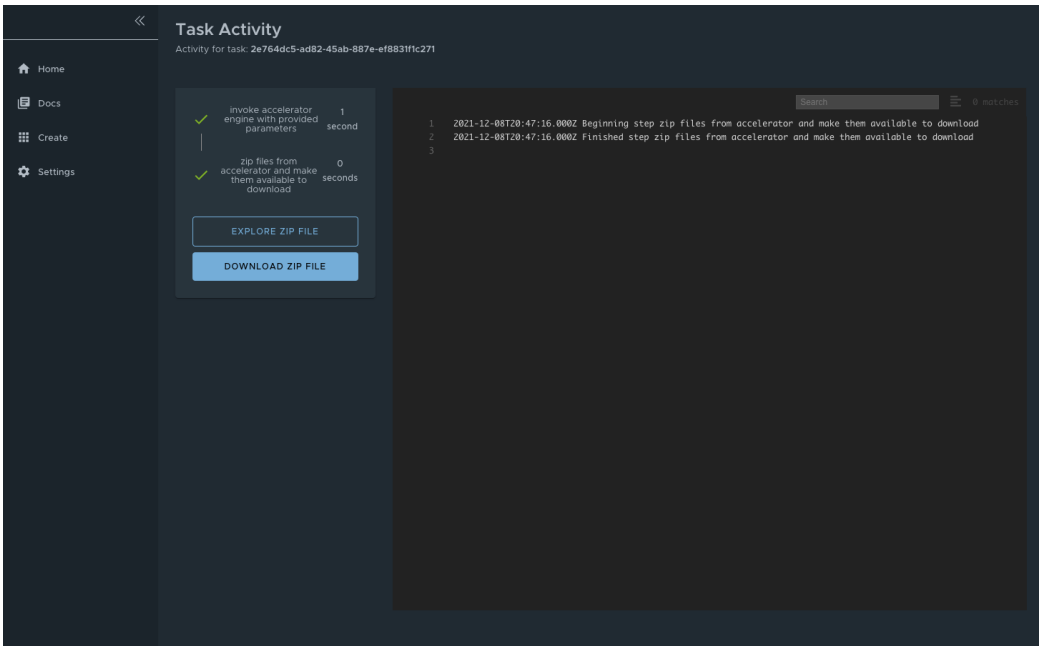


2. Click **EXPLORE** to open the **Explore Project** page and view the project before it is generated.
3. After configuring your project, click **NEXT STEP** to see the project summary page.
4. Review the values you specified for the configurable options.
5. Click **BACK** to make more changes, if necessary. Otherwise, proceed to create the project.

Create the project

To create the project:

1. Click **Create** to start generating your project. See the progress on the **Task Activity** page. A detailed log is displayed on the right.

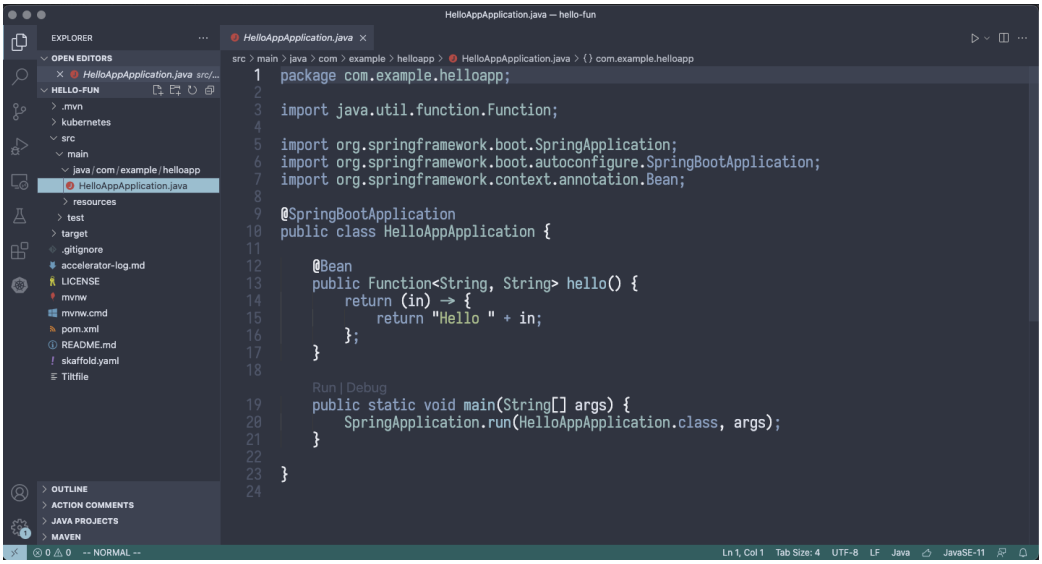


2. After the project is generated, click **EXPLORE ZIP FILE** to open the **Explore Project** page to verify configuration.
3. Click **DOWNLOAD ZIP FILE** to download the project in a ZIP file.

Develop your code

To develop your code:

1. Expand the ZIP file.
2. Open the project in your integrated development environment (IDE).



Next steps

To learn more about Application Accelerator for VMware Tanzu, see the Application Accelerator documentation.

Install Application Accelerator

This topic tells you how to install Application Accelerator from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Application Accelerator. For more information about profiles, see About Tanzu Application Platform components and profiles.

Prerequisites

Before installing Application Accelerator:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- Install Flux SourceController on the cluster. See Install Flux CD Source Controller.
- Install Source Controller on the cluster. See Install Source Controller.

Install

To install Application Accelerator:

1. List version information for the package by running:

   ```
   tanzu package available list accelerator.apps.tanzu.vmware.com --namespace tap-install
   ```

   For example:

   ```
   $ tanzu package available list accelerator.apps.tanzu.vmware.com --namespace tap-install

   - Retrieving package versions for accelerator.apps.tanzu.vmware.com...
     NAME                                VERSION  RELEASED-AT
     accelerator.apps.tanzu.vmware.com  1.4.0    2022-12-08 12:00:00 -0500 EST
   ```

2. (Optional) View the changes you can make to the default installation settings by running:

   ```
   tanzu package available get accelerator.apps.tanzu.vmware.com/VERSION-NUMBER \
     --values-schema \
     --namespace tap-install
   ```

   Where `VERSION-NUMBER` is the version of the Application Accelerator package listed earlier.

   For example:

   ```
   $ tanzu package available get accelerator.apps.tanzu.vmware.com/1.4.0 \
       --values-schema \
       --namespace tap-install
   ```

For more information about values schema options, see the properties listed in Configure properties and resource use later.

3. Create a file named `app-accelerator-values.yaml` using the following example code:

```
server:
  service_type: "LoadBalancer"
  watched_namespace: "accelerator-system"
samples:
  include: true
```

4. Edit the values in your `app-accelerator-values.yaml` if needed, or leave the default values. You can add values you want from Configure properties and resource use.

5. Install the package by running:

```
tanzu package install app-accelerator \
  --package accelerator.apps.tanzu.vmware.com \
  --version VERSION-NUMBER \
  --namespace tap-install \
  --values-file app-accelerator-values.yaml
```

Where `VERSION-NUMBER` is the version of the Application Accelerator package listed earlier.

For example:

```
$ tanzu package install app-accelerator \
    --package accelerator.apps.tanzu.vmware.com \
    --version 1.4.0 \
    --namespace tap-install \
    --values-file app-accelerator-values.yaml

- Installing package 'accelerator.apps.tanzu.vmware.com'
| Getting package metadata for 'accelerator.apps.tanzu.vmware.com'
| Creating service account 'app-accelerator-tap-install-sa'
| Creating cluster admin role 'app-accelerator-tap-install-cluster-role'
| Creating cluster role binding 'app-accelerator-tap-install-cluster-rolebindin
g'
| Creating secret 'app-accelerator-tap-install-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'app-accelerator' in namespace 'tap-install'
```

6. Verify the package install by running:

```
tanzu package installed get app-accelerator -n tap-install
```

For example:

```
$ tanzu package installed get app-accelerator -n tap-install

| Retrieving installation details for cc...
NAME:                    app-accelerator
PACKAGE-NAME:            accelerator.apps.tanzu.vmware.com
PACKAGE-VERSION:         1.4.0
STATUS:                  Reconcile succeeded
CONDITIONS:              [{ReconcileSucceeded True  }]
USEFUL-ERROR-MESSAGE:
```

Verify that `STATUS` is `Reconcile succeeded`.

7. To see the IP address for the Application Accelerator API when the `server.service_type` is set to `LoadBalancer`, run:

   ```
   kubectl get service -n accelerator-system
   ```

   This lists an external IP address for use with the `--server-url` Tanzu CLI flag for the Accelerator plug-in `generate` & `generate-from-local` command.

For how to troubleshoot installation issues, see Troubleshoot Application Accelerator.

Configure properties and resource use

When you install the Application Accelerator, you can configure the following optional properties from within your `app-accelerator-values.yaml` configuration file:

Property	Default	Description
`registry.secret_ref`	`registry.tanzu.vmware.com`	The secret used for accessing the registry where the App-Accelerator images are located
`server.service_type`	`ClusterIP`	The service type for the acc-ui-server service including LoadBalancer, NodePort, or ClusterIP
`server.watched_namespace`	`accelerator-system`	The namespace the server watches for accelerator resources
`server.engine_invocation_url`	`http://acc-engine.accelerator-system.svc.cluster.local/invocations`	The URL to use for invoking the accelerator engine
`engine.service_type`	`ClusterIP`	The service type for the acc-engine service including LoadBalancer, NodePort, or ClusterIP
`engine.max_direct_memory_size`	`32M`	The maximum size for the Java -XX:MaxDirectMemorySize setting
`samples.include`	`True`	Option to include the bundled sample Accelerators in the installation
`ingress.include`	`False`	Option to include the ingress configuration in the installation
`ingress.enable_tls`	`False`	Option to include TLS for the ingress configuration
`domain`	`tap.example.com`	Top-level domain to use for ingress configuration, default is `shared.ingress_domain`
`tls.secret_name`	`tls`	The name of the secret
`tls.namespace`	`tanzu-system-ingress`	The namespace for the secret
`telemetry.retain_invocation_events_for_no_days`	`30`	The number of days to retain recorded invocation events resources
`telemetry.record_invocation_events`	`true`	The system records each engine invocation when generating files for an accelerator?
`git_credentials.secret_name`	`git-credentials`	The name to use for the secret storing Git credentials for accelerators
`git_credentials.username`	`null`	The user name to use in secret storing Git credentials for accelerators
`git_credentials.password`	`null`	The password to use in secret storing Git credentials for accelerators

Property	Default	Description
`git_credentials.ca_file`	`null`	The CA certificate data to use in secret storing Git credentials for accelerators
`managed_resources.enable`	`false`	Whether to enable the App used to control managed accelerator resources
`managed_resources.git.url`	none	Required if managed_resources are enabled. Git repository URL containing manifests for managed accelerator resources
`managed_resources.git.ref`	`origin/main`	Required if managed_resources are enabled. Git ref to use for repository containing manifests for managed accelerator resources
`managed_resources.git.sub_path`	`null`	Git subPath to use for repository containing manifests for managed accelerator resources
`managed_resources.git.secret_ref`	`git-credentials`	Secret name to use for repository containing manifests for managed accelerator resources

VMware recommends that you do not override the default setting for `registry.secret_ref`, `server.engine_invocation_url`, or `engine.service_type`. These properties are only used to configure non-standard installations.

The following table is the resource use configurations for the components of Application Accelerator.

Component	Resource requests	Resource limits
acc-controller	CPU: 100m memory: 20Mi	CPU: 100m memory: 30Mi
acc-server	CPU: 100m memory:20Mi	CPU: 100m memory: 30Mi
acc-engine	CPU: 500m memory: 1Gi	CPU: 500m memory: 2Gi

Create an Application Accelerator Git repository during project creation

This topic tells you how to enable and use GitHub repository creation in the Application Accelerator plug-in of Tanzu Application Platform GUI (commonly called TAP GUI).

Overview

The Application Accelerator plug-in uses the Backstage GitHub provider integration and the authentication mechanism to retrieve an access token. Then it can interact with the provider API to create GitHub repositories.

Supported Providers

The supported Git providers are GitHub and GitLab.

Note

To create a repository in a self-hosted GitLab, you must add a custom GitLab integration in `tap-values.yaml` as described in the Full Profile sample.

Configure

The following steps describe an example configuration that uses GitHub:

1. Create an **OAuth App** in GitHub based on the configuration described in this Backstage documentation. GitHub Apps are not supported. For more information about creating an OAuth App in GitHub, see the GitHub documentation.

   These values appear in your `app-config.yaml` or `app-config.local.yaml` for local development. For example:

   ```
   auth:
    environment: development
    providers:
      github:
        development:
          clientId: GITHUB-CLIENT-ID
          clientSecret: GITHUB-CLIENT-SECRET
   ```

2. Add a GitHub integration in your `app-config.yaml` configuration. For example:

   ```
   app_config:
    integrations:
      github:
        - host: github.com
   ```

   For more information, see the Backstage documentation.

(Optional) Deactivate Git repository creation

As of Tanzu Application Platform v1.5, you can deactivate Git repository creation by setting the property `customize.features.accelerators.gitRepoCreation` to `false` in `tap-values.yaml`. This also deactivates Git repository creation in the Application Accelerator extension for VS Code.

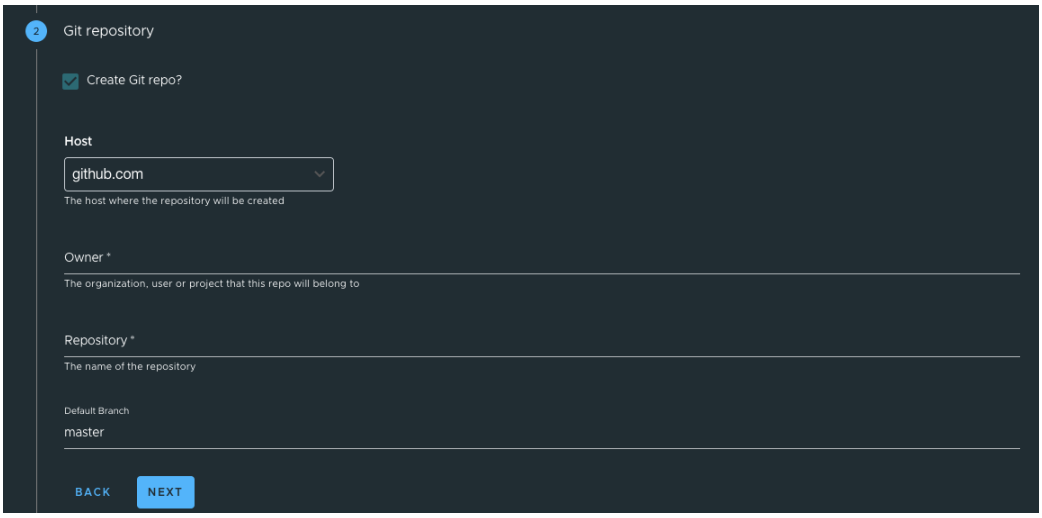
See the following example configuration for deactivating Git repository creation:

```
app_config:
  customize:
    features:
      accelerators:
        gitRepoCreation: false
```

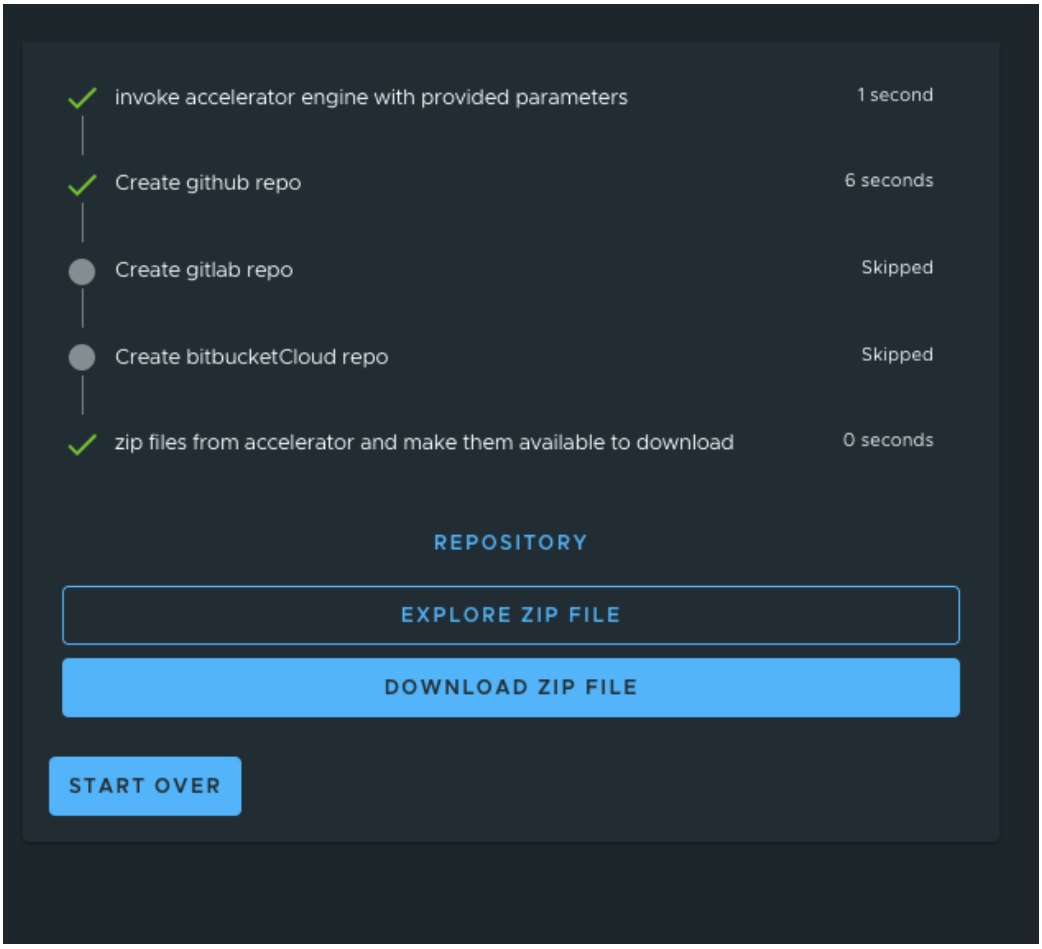
Create a Project

To create a project:

1. Go to Tanzu Application Platform GUI, access the Accelerators section, and then select an accelerator. The accelerator form now has a second step named **Git repository**.
2. Fill in the accelerator options and click **Next**.
3. Select the **Create Git repo?** check box.
4. Fill in the **Owner**, **Repository**, and **Default Branch** text boxes.



5. After entering the repository name, a dialog box appears that requests GitHub credentials. Log in and then click **Next**.
6. Click **GENERATE ACCELERATOR**. A link to the repository location appears.



API documentation plug-in in Tanzu Application Platform GUI

This topic gives you an overview of the API documentation plug-in of Tanzu Application Platform GUI (commonly called TAP GUI). For more information, see Get started with the API documentation plug-in.

Overview

The API documentation plug-in provides a standalone list of APIs that can be connected to components and systems of the Tanzu Application Platform GUI software catalog.

Each API entity can reflect the components that provide that API and the list of components that are consumers of that API. Also, an API entity can be associated with systems and appear on the system diagram. To show this dependency, make the `spec.providesApis:` and `spec.consumesApis:` sections of the component definition files reference the name of the API entity.

Here's a sample of how you can add `providesApis` and `consumesApis` to an existing component's catalog definition, linking them together.

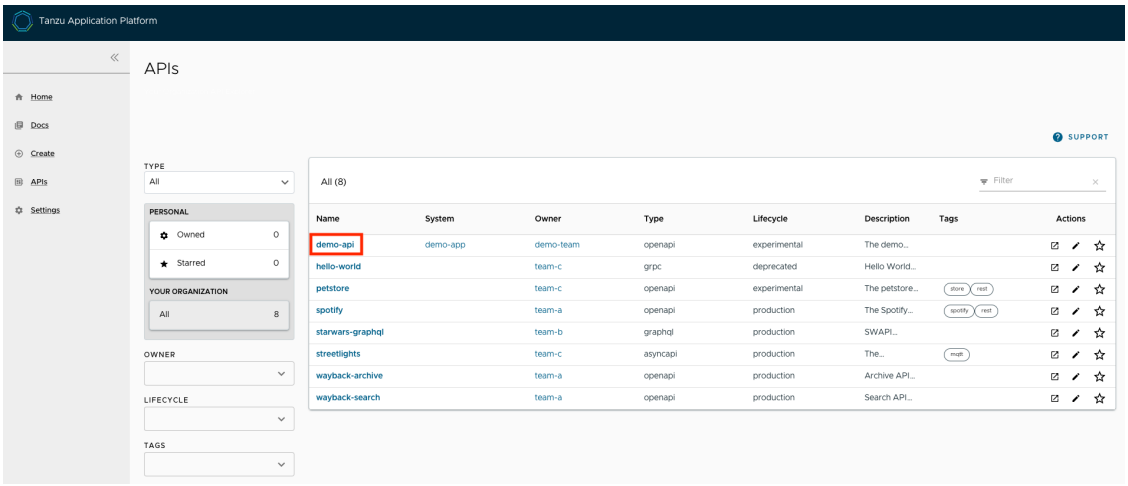
```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: example-component
  description: Example Component
spec:
  type: service
  lifecycle: experimental
  owner: team-a
  system: example-system
  providesApis: # list of APIs provided by the Component
    - example-api-1
  consumesApis: # list of APIs consumed by the Component
    - example-api-2
```

For more information about the structure of the definition file for an API entity, see the Backstage Kind: API documentation. For more information about the API documentation plug-in, see the Backstage API documentation in GitHub.

Use the API documentation plug-in

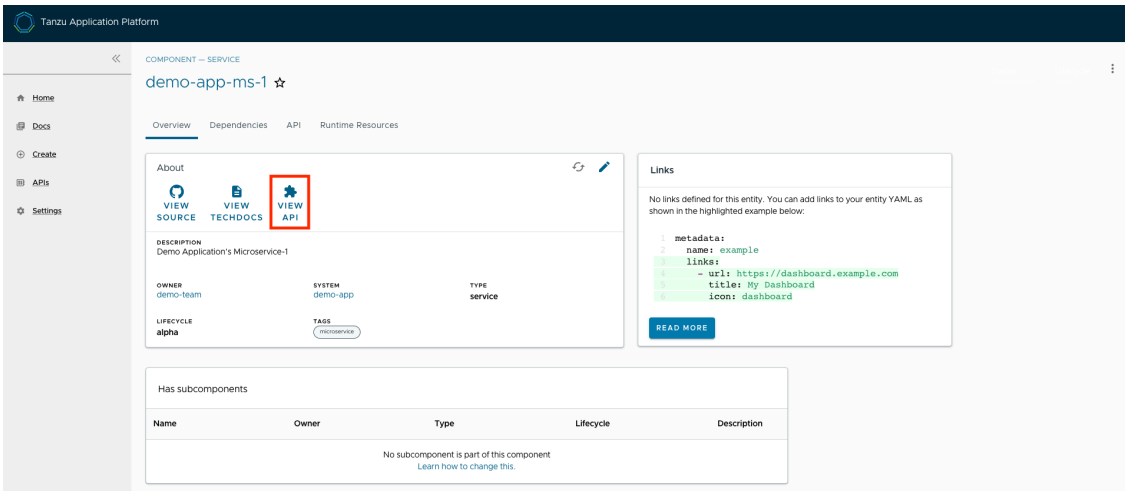
The API documentation plug-in is part of Tanzu Application Platform GUI.

The first way to use the API documentation plug-in is API-first. Click **APIs** in the left navigation pane of Tanzu Application Platform GUI. This opens the **API catalog page**.

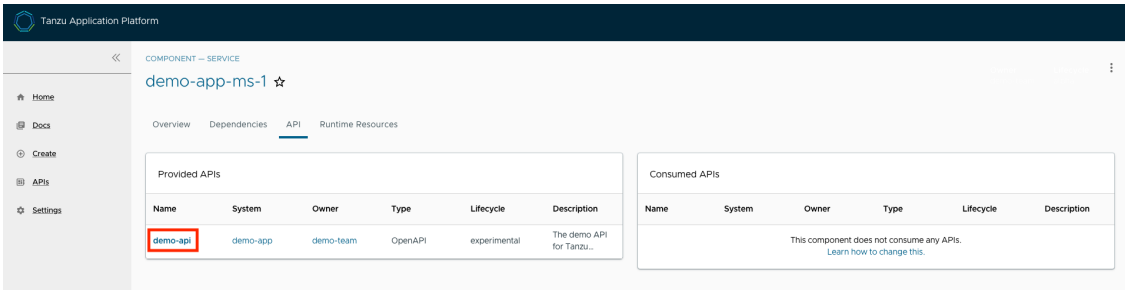


On that page, you can view all the APIs already registered in the catalog regardless of whether they are associated with components or systems.

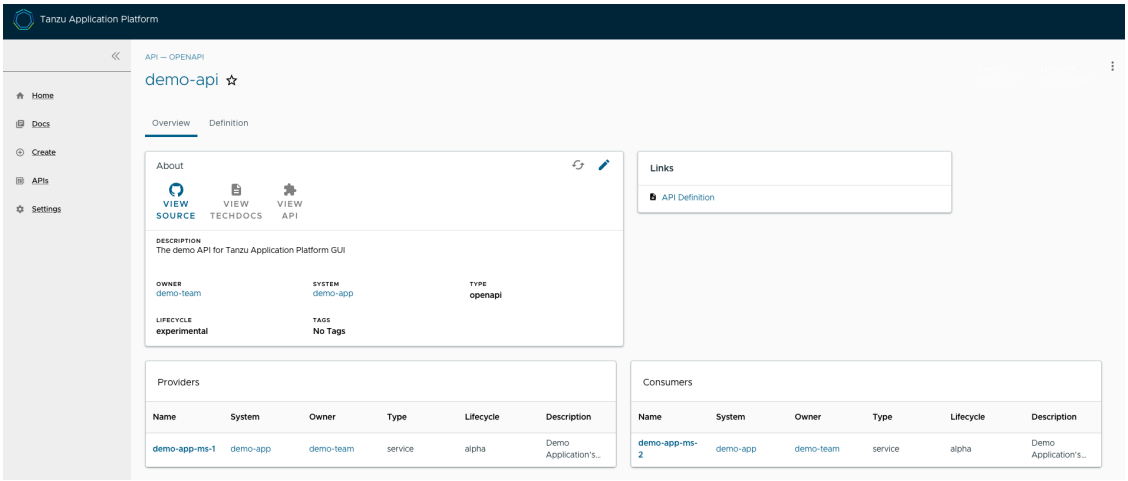
The second way to use the API documentation plug-in is to use components and systems of the software catalog, listed on the home page of Tanzu Application Platform GUI. If there is an API entity associated with the selected component or system, the **VIEW API** icon is active.



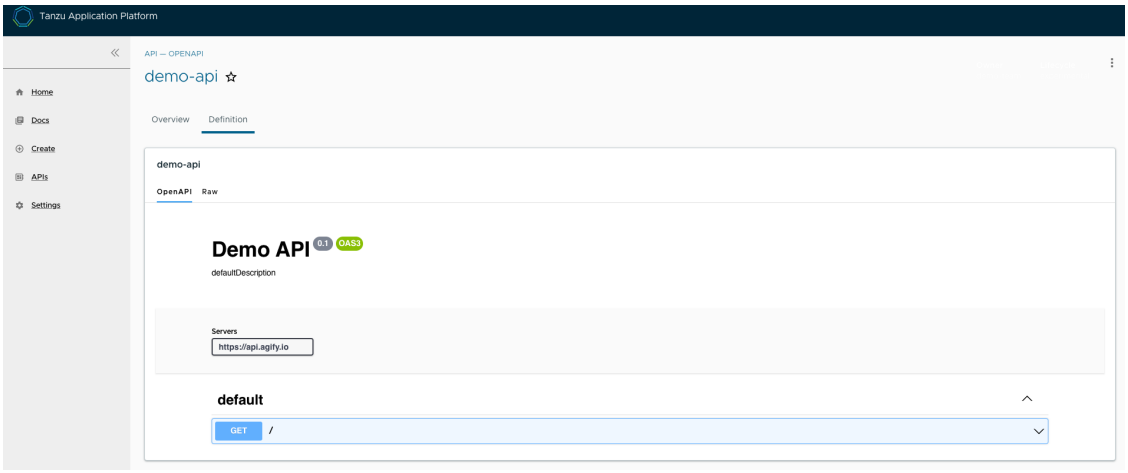
The **VIEW API** tab displays which APIs are being consumed by a component and which APIs are being provided by the component.



Clicking on the API itself takes you to the catalog entry for the API, which the Kind type listed in the upper-left corner denotes. Every API entity has a title and short description, including a reference to the team that owns the definition of that API and the software catalog objects that are connected to it.



Select the **Definition** tab on the top of the API page to see the definition of that API in human-readable and machine-readable format.



The API documentation plug-in supports the following API formats:

- OpenAPI 2 & 3
- AsyncAPI
- GraphQL
- Plain (to support any other format)

Create a new API entry

You can create a new API entry manually or automatically.

Manually create a new API entry

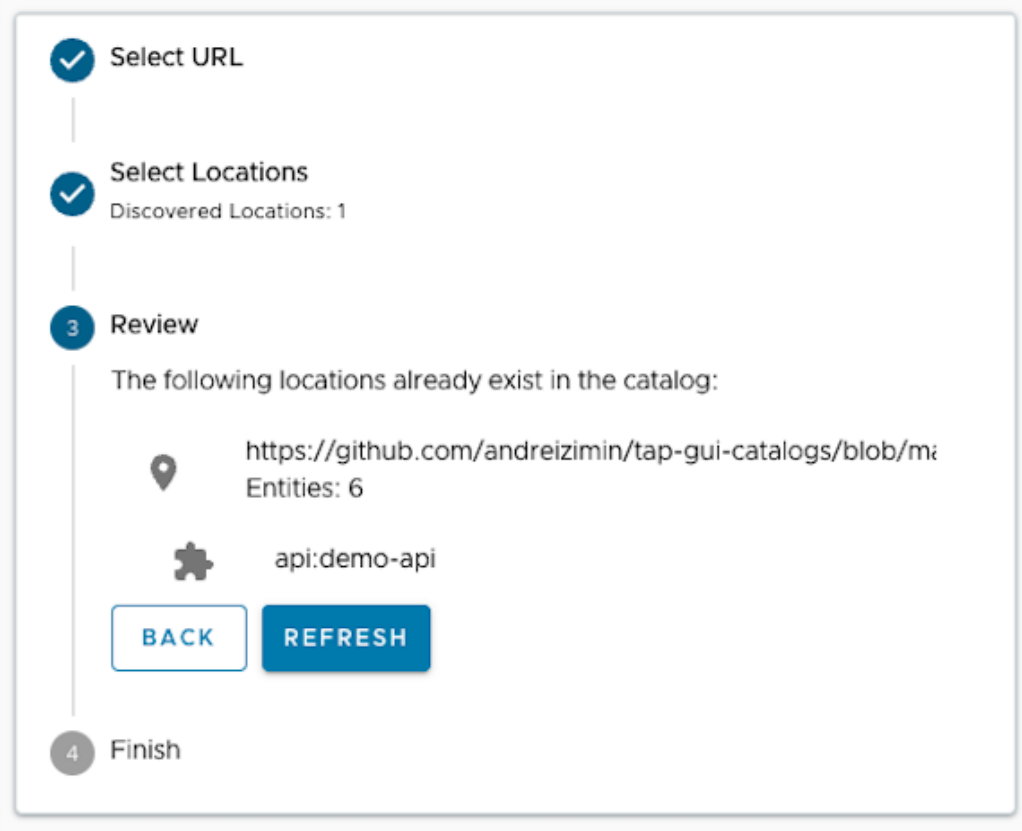
Manually creating a new API entity is similar to registering any other software catalog entity. To manually create a new API entity:

1. Click the **Home** button on the left navigation pane to access the home page of Tanzu Application Platform GUI.
2. Click **REGISTER ENTITY**.
3. **Register an existing component** prompts you to type a repository URL. Paste the link to the `catalog-info.yaml` file of your choice that contains the definition of your API entity. For example, you can copy the following YAML content and save it as `catalog-info.yaml` on a Git repository of your choice.

```
apiVersion: backstage.io/v1alpha1
kind: API
metadata:
name: demo-api
description: The demo API for Tanzu Application Platform GUI
links:
  - url: https://api.agify.io
    title: API Definition
    icon: docs
spec:
type: openapi
lifecycle: experimental
owner: demo-team
system: demo-app # Or specify system name of your choice
definition: |
  openapi: 3.0.1
  info:
    title: defaultTitle
    description: defaultDescription
```

```
  version: '0.1'
servers:
  - url: https://api.agify.io
paths:
  /:
    get:
      description: Auto generated using Swagger Inspector
      parameters:
        - name: name
          in: query
          schema:
            type: string
          example: type_any_name
      responses:
        '200':
          description: Auto generated using Swagger Inspector
          content:
            application/json; charset=utf-8:
              schema:
                type: string
              examples: {}
```

4. Click **ANALYZE** and then review the catalog entities to be added.



5. Click **IMPORT**.
6. Click **APIs** on the left navigation pane to view entries on the **API** page.

Automatically create a new API entry

Tanzu Application Platform v1.3 introduced a feature called **API Auto Registration** that can automatically register your APIs. For more information, see API Auto Registration.

API documentation plug-in in Tanzu Application Platform GUI

This topic gives you an overview of the API documentation plug-in of Tanzu Application Platform GUI (commonly called TAP GUI). For more information, see Get started with the API documentation plug-in.

Overview

The API documentation plug-in provides a standalone list of APIs that can be connected to components and systems of the Tanzu Application Platform GUI software catalog.

Each API entity can reflect the components that provide that API and the list of components that are consumers of that API. Also, an API entity can be associated with systems and appear on the system diagram. To show this dependency, make the `spec.providesApis:` and `spec.consumesApis:` sections of the component definition files reference the name of the API entity.

Here's a sample of how you can add `providesApis` and `consumesApis` to an existing component's catalog definition, linking them together.

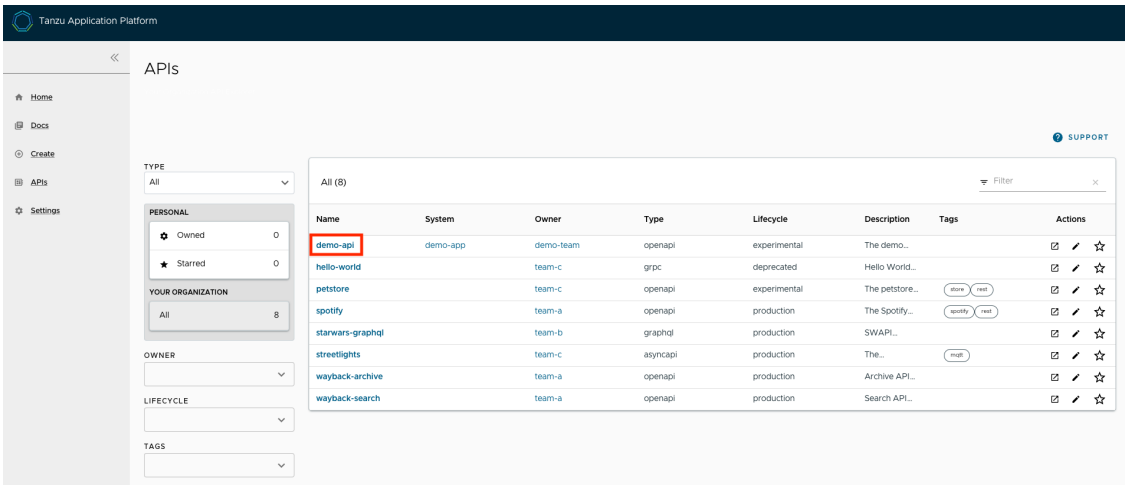
```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: example-component
  description: Example Component
spec:
  type: service
  lifecycle: experimental
  owner: team-a
  system: example-system
  providesApis: # list of APIs provided by the Component
    - example-api-1
  consumesApis: # list of APIs consumed by the Component
    - example-api-2
```

For more information about the structure of the definition file for an API entity, see the Backstage Kind: API documentation. For more information about the API documentation plug-in, see the Backstage API documentation in GitHub.

Use the API documentation plug-in

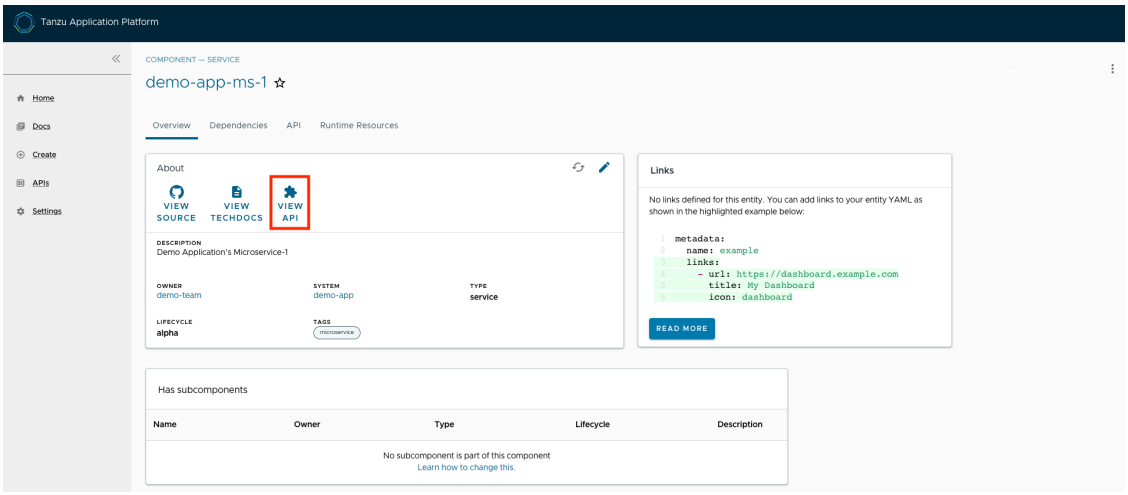
The API documentation plug-in is part of Tanzu Application Platform GUI.

The first way to use the API documentation plug-in is API-first. Click **APIs** in the left navigation pane of Tanzu Application Platform GUI. This opens the **API catalog page**.

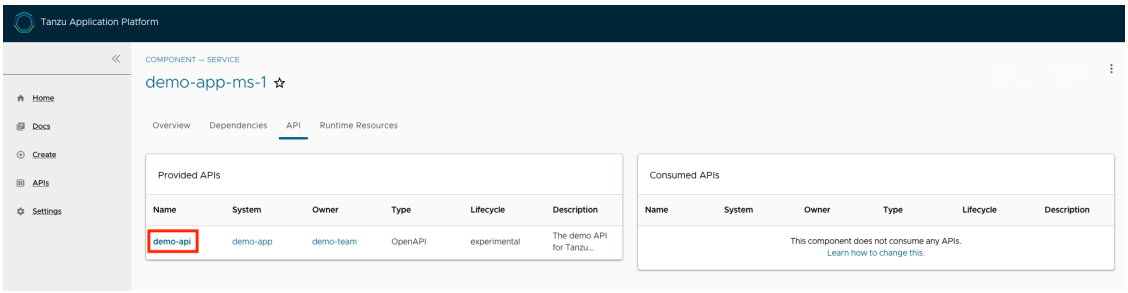


On that page, you can view all the APIs already registered in the catalog regardless of whether they are associated with components or systems.

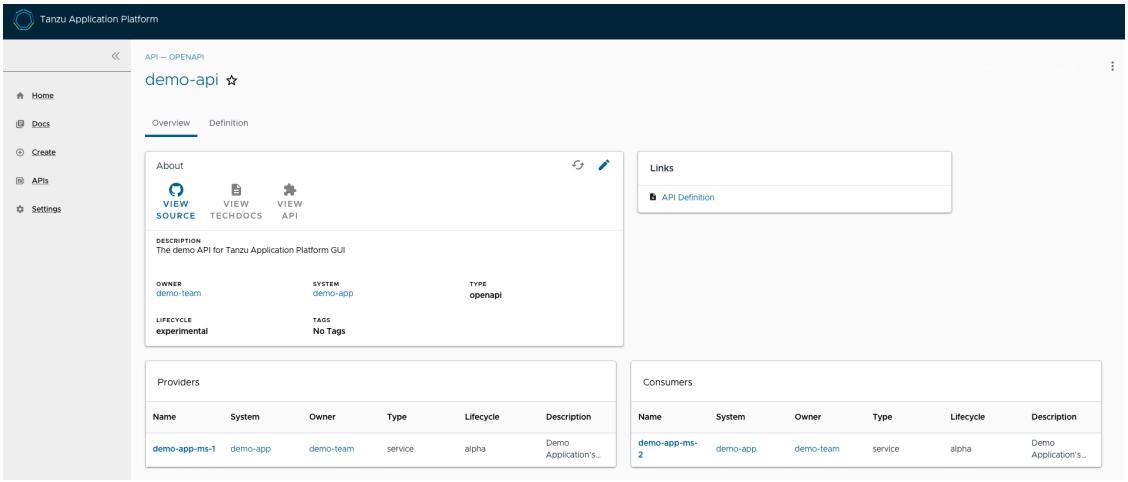
The second way to use the API documentation plug-in is to use components and systems of the software catalog, listed on the home page of Tanzu Application Platform GUI. If there is an API entity associated with the selected component or system, the **VIEW API** icon is active.



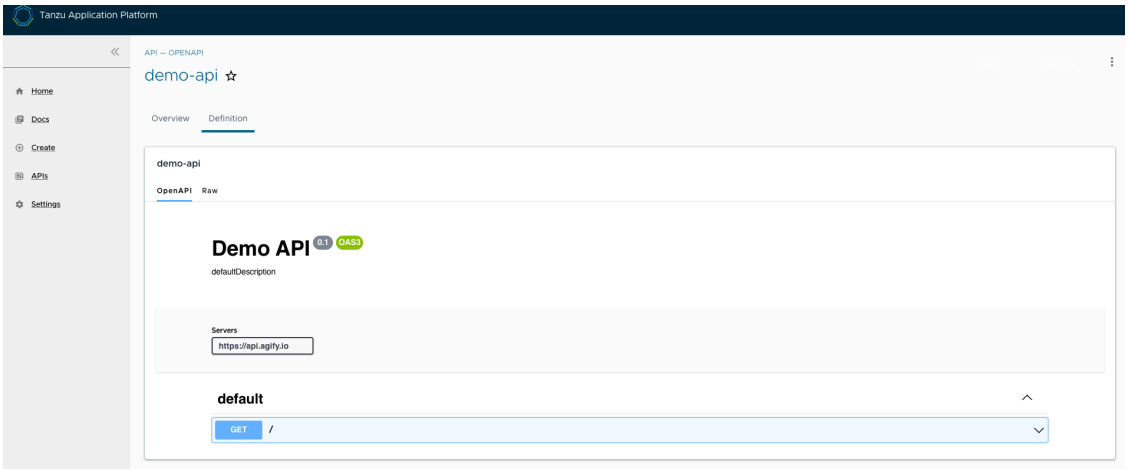
The **VIEW API** tab displays which APIs are being consumed by a component and which APIs are being provided by the component.



Clicking on the API itself takes you to the catalog entry for the API, which the Kind type listed in the upper-left corner denotes. Every API entity has a title and short description, including a reference to the team that owns the definition of that API and the software catalog objects that are connected to it.



Select the **Definition** tab on the top of the API page to see the definition of that API in human-readable and machine-readable format.



The API documentation plug-in supports the following API formats:

- OpenAPI 2 & 3
- AsyncAPI
- GraphQL
- Plain (to support any other format)

Create a new API entry

You can create a new API entry manually or automatically.

Manually create a new API entry

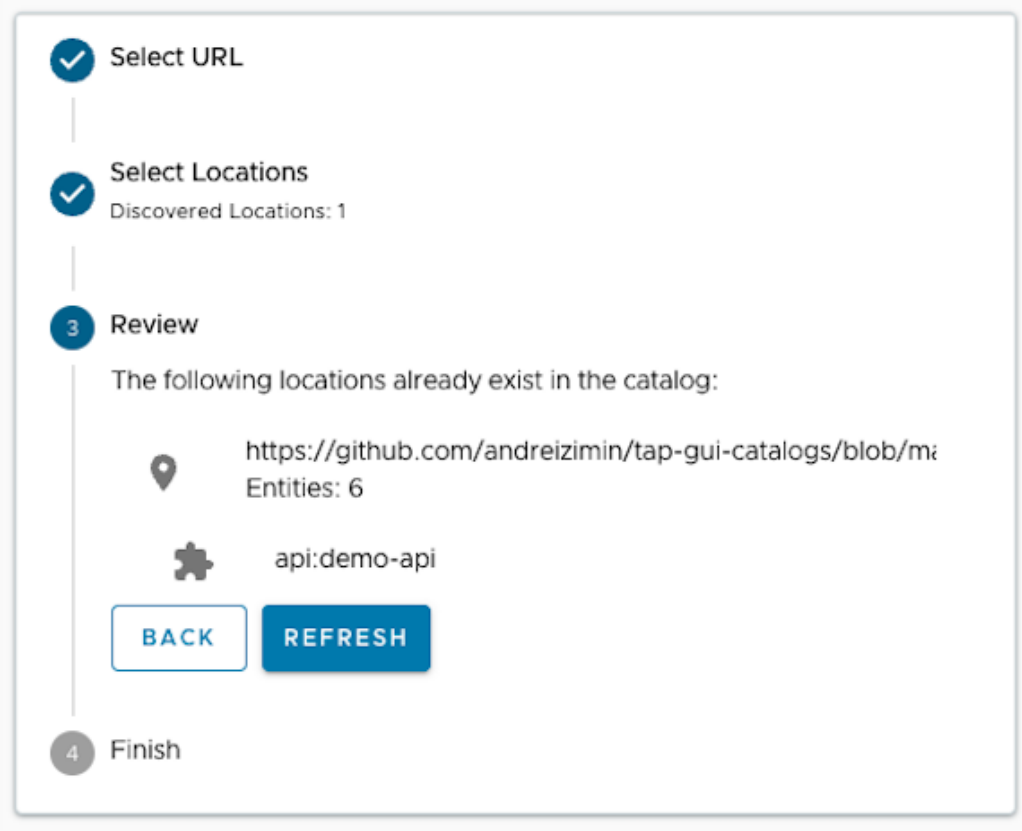
Manually creating a new API entity is similar to registering any other software catalog entity. To manually create a new API entity:

1. Click the **Home** button on the left navigation pane to access the home page of Tanzu Application Platform GUI.
2. Click **REGISTER ENTITY**.
3. **Register an existing component** prompts you to type a repository URL. Paste the link to the `catalog-info.yaml` file of your choice that contains the definition of your API entity. For example, you can copy the following YAML content and save it as `catalog-info.yaml` on a Git repository of your choice.

```
apiVersion: backstage.io/v1alpha1
kind: API
metadata:
name: demo-api
description: The demo API for Tanzu Application Platform GUI
links:
  - url: https://api.agify.io
    title: API Definition
    icon: docs
spec:
type: openapi
lifecycle: experimental
owner: demo-team
system: demo-app # Or specify system name of your choice
definition: |
  openapi: 3.0.1
  info:
    title: defaultTitle
    description: defaultDescription
```

```
  version: '0.1'
servers:
  - url: https://api.agify.io
paths:
  /:
    get:
      description: Auto generated using Swagger Inspector
      parameters:
        - name: name
          in: query
          schema:
            type: string
          example: type_any_name
      responses:
        '200':
          description: Auto generated using Swagger Inspector
          content:
            application/json; charset=utf-8:
              schema:
                type: string
              examples: {}
```

4. Click **ANALYZE** and then review the catalog entities to be added.



5. Click **IMPORT**.
6. Click **APIs** on the left navigation pane to view entries on the **API** page.

Automatically create a new API entry

Tanzu Application Platform v1.3 introduced a feature called **API Auto Registration** that can automatically register your APIs. For more information, see API Auto Registration.

Get started with the API documentation plug-in

This topic tells you how to get started with the API documentation plug-in in Tanzu Application Platform GUI (commonly called TAP GUI).

API entries

This section describes API entities, how to add them, and how to update them.

About API entities

The list of API entities is visible on the left side navigation pane of Tanzu Application Platform GUI. It is also visible on the Overview page of specific components on the home page. APIs are a definition of the interface between components.

Their definition is provided in raw machine-readable and human-readable formats. For more information, see the API plug-in documentation.

Add a demo API entity to the Tanzu Application Platform GUI software catalog

To add a demo API entity and its related Catalog objects, follow the steps used for registering any other software catalog entity:

1. Go to the Home page of Tanzu Application Platform GUI by clicking **Home** on the left-side navigation pane.
2. Click **REGISTER ENTITY**.
3. In the repository URL text box, type the link to the `catalog-info.yaml` file of your choice or use the following sample definition.
4. Save this code block as `catalog-info.yaml`.
5. Upload it to the Git repository of your choice and copy the link to `catalog-info.yaml`. This demo setup includes a domain named `demo-domain` with a single system named `demo-system`. This systems consists of two microservices (`demo-app-ms-1` and `demo-app-ms-1`) and one API named `demo-api` that `demo-app-ms-1` provides and that `demo-app-ms-2` consumes.

```
apiVersion: backstage.io/v1alpha1
kind: Domain
metadata:
  name: demo-domain
  description: Demo Domain for Tanzu Application Platform
  annotations:
    'backstage.io/techdocs-ref': dir:.
spec:
  owner: demo-team

---

apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-app-ms-1
  description: Demo Application's Microservice-1
  tags:
    - microservice
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=demo-a
pp-ms-1'
    'backstage.io/techdocs-ref': dir:.
spec:
  type: service
```

```
  providesApis:
   - demo-api
  lifecycle: alpha
  owner: demo-team
  system: demo-app

---

apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
  name: demo-app-ms-2
  description: Demo Application's Microservice-2
  tags:
    - microservice
  annotations:
    'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=demo-a
pp-ms-2'
    'backstage.io/techdocs-ref': dir:.
spec:
  type: service
  consumesApis:
   - demo-api
  lifecycle: alpha
  owner: demo-team
  system: demo-app

---

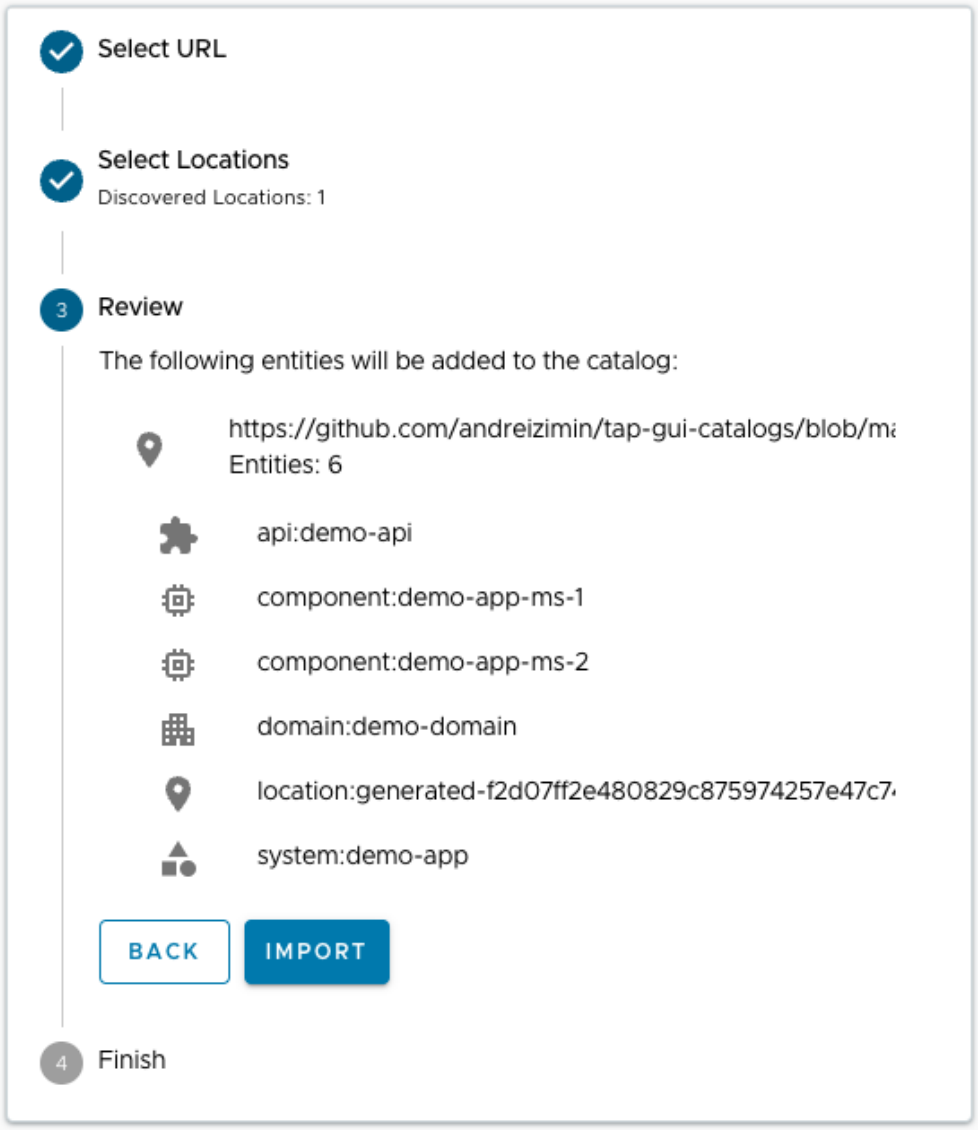
apiVersion: backstage.io/v1alpha1
kind: System
metadata:
  name: demo-app
  description: Demo Application for Tanzu Application Platform
  annotations:
    'backstage.io/techdocs-ref': dir:.
spec:
  owner: demo-team
  domain: demo-domain

---

apiVersion: backstage.io/v1alpha1
kind: API
metadata:
  name: demo-api
  description: The demo API for Tanzu Application Platform GUI
  links:
    - url: https://api.agify.io
      title: API Definition
      icon: docs
spec:
  type: openapi
  lifecycle: experimental
  owner: demo-team
  system: demo-app # Or specify system name of your choice
  definition: |
    openapi: 3.0.1
    info:
      title: Demo API
      description: defaultDescription
      version: '0.1'
    servers:
      - url: https://api.agify.io
    paths:
      /:
```

```
    get:
      description: Auto generated using Swagger Inspector
      parameters:
        - name: name
          in: query
          schema:
            type: string
          example: type_any_name
      responses:
        '200':
          description: Auto generated using Swagger Inspector
          content:
            application/json; charset=utf-8:
              schema:
                type: string
              examples: {}
```

6. Paste the link to `catalog-info.yaml` and click **ANALYZE**.
7. Review the catalog entities and click **IMPORT**.



8. Go to the **API** page by clicking **APIs** on the left side navigation pane. The catalog changes and entries are visible for further inspection. If you select the system **demo-app**, the diagram appears as follows:

Update your demo API entry

To update your demo API entity, click on **demo-api** from the list of available APIs in your software catalog and click the **Edit** icon on the Overview page.

It opens the source `catalog-info.yaml` file that you can edit. For example, you can change the `spec.paths.parameters.example` from `type_any_name` to `Tanzu` and then save your changes.

```
![Screenshot of the overview of demo dash api. The edit button on the card labeled Abo
ut is framed in red.](../images/api-plugin-9.png)

It opens the source `catalog-info.yaml` file that you can edit. For example, you can c
hange the
`spec.paths.parameters.example` from `type_any_name` to `Tanzu` and then save your cha
nges.
```

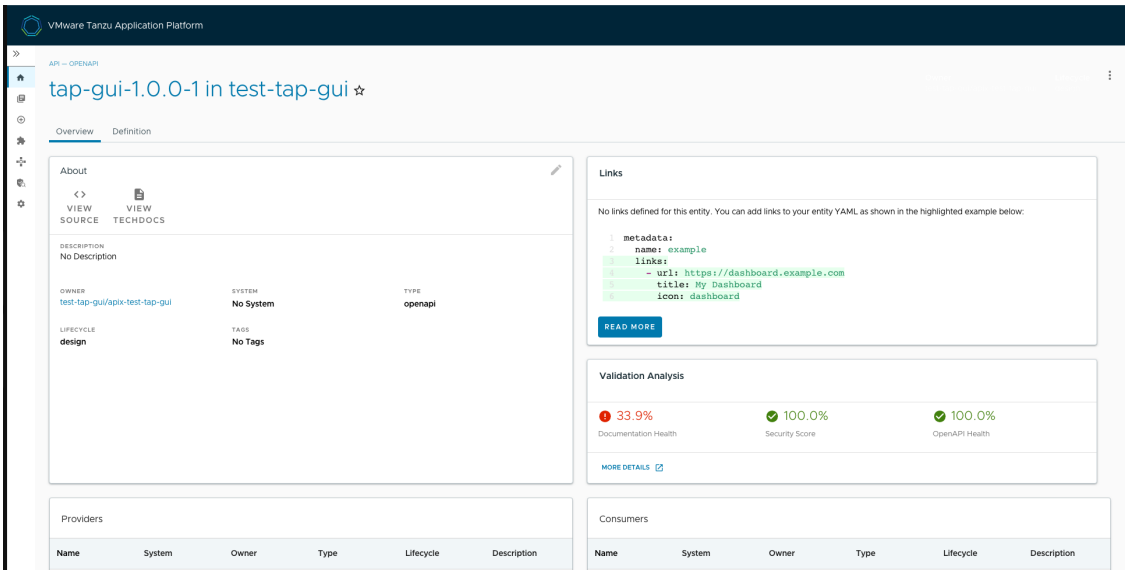
After making any edits, Tanzu Application Platform GUI re-renders the API entry with the next refresh cycle.

Validation Analysis of API specifications

This section describes the Validation Analysis card, the data format needed to populate the card, and how to get automatic scores for your OpenAPI entities.

About the Validation Analysis card

When viewing entities of the kind `API` on the Overview tab, you see the Validation Analysis card that displays the health of an API through various scoring parameters.



To display the health scores, an API entity must contain the following metadata structure:

```
apiVersion: backstage.io/v1alpha1
kind: API
metadata:
  name: NAME
  description: DESCRIPTION
  apiscores:
    scores:
    - id: documentationReport
      label: "Documentation Health"
      value: 34.375
      valueType: percentage
      status: failed
    - id: securityReport
      label: "Security Score"
      value: 70.0
      valueType: percentage
      status: warning
    - id: openApiHealthReport
      label: "OpenAPI Health"
      value: 89.0625
      valueType: percentage
      status: passed
    scoreDetailsURL:  VALIDATION-REPORT-URL-FOR-MORE-DETAILS
# Other API Entity parameters
```

If an API entity follows this schema, the Validation Analysis card displays helpful information about the API.

```
    - id:        # Unique ID
      label:     # Descriptive label displayed as a title over the numerical value
      value:     # Any number value
      valueType: # One of the types (percentage or other). Displays the % symbol or di
splays nothing.
      status:    # One of the statuses (passed, warning, or failed). Displays the numb
er in green, yellow, or red.
```

Automatic OpenAPI specification validation

To receive automatic validation analysis for OpenAPI specifications by using API Validation Scoring:

1. Install API Validation and Scoring.

2. Use API Auto Registration or API Validation Scoring Design GitOps to automatically generate the API entities in Tanzu Application Platform GUI.

The automatic scoring cannot score or replace API entities created through other methods, such as regular GitOps or manual registration. You might see the following message signaling that the OpenAPI specification was registered with regular GitOps methods or manual registration.

```
Validation analysis is currently unavailable for APIs registered via TAP GUI without b
eing attached \
to a workload.
```

Security Analysis in Tanzu Application Platform GUI

This topic tells you about the Security Analysis plug-in in Tanzu Application Platform GUI (commonly called TAP GUI).

Overview

The Security Analysis plug-in summarizes vulnerability data across all workloads running in Tanzu Application Platform, enabling faster identification and remediation of CVEs.

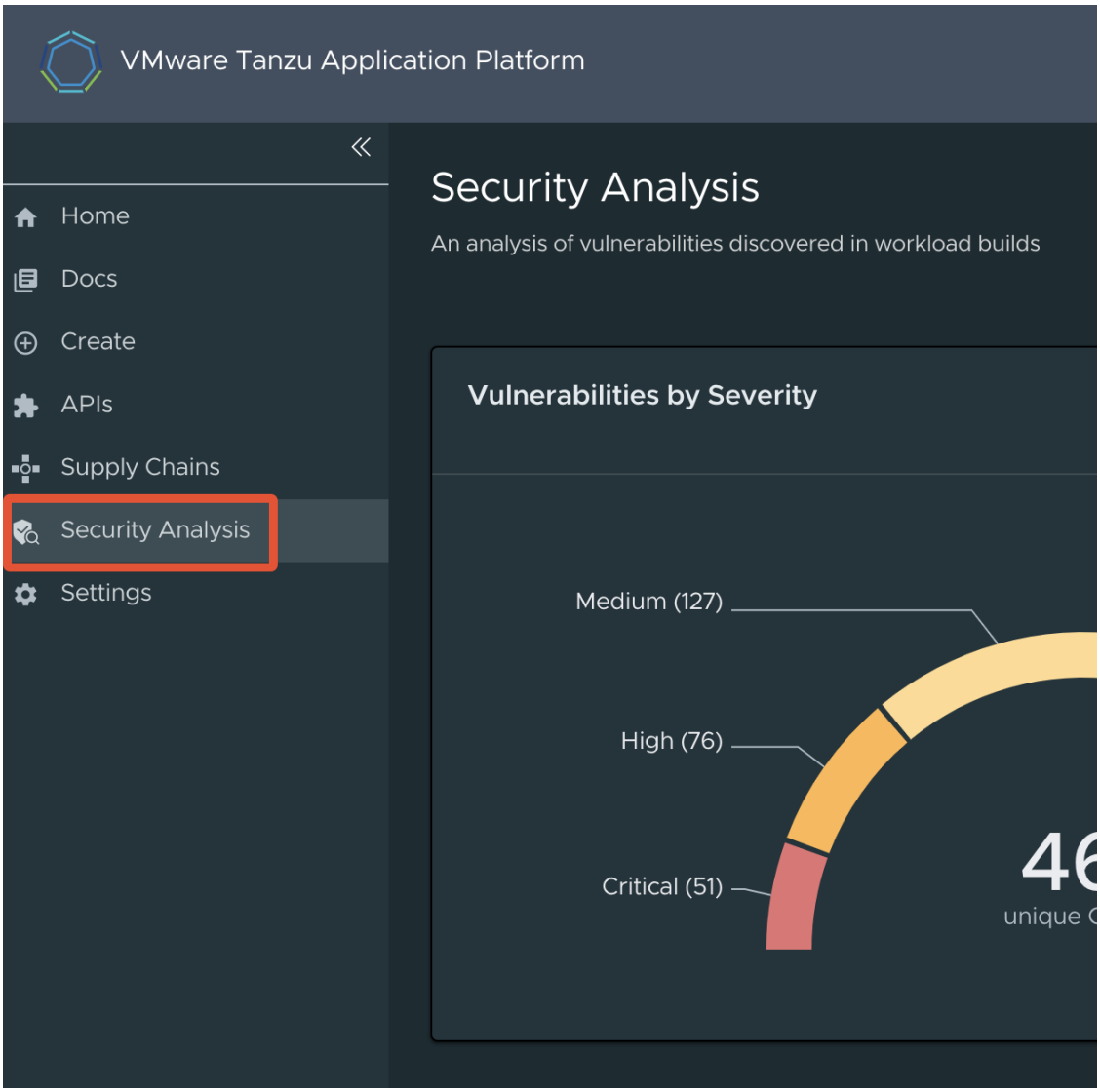
Installing and configuring

The Security Analysis plug-in is installed by default. It is tightly coupled with the Supply Chain Choregrapher plug-in. After installing and configuring the Supply Chain Choreographer GUI plug-in, there is no additional configuration needed for the Security Analysis plug-in.

The Security Analysis plug-in is part of the Tanzu Application Platform Full and View profiles.

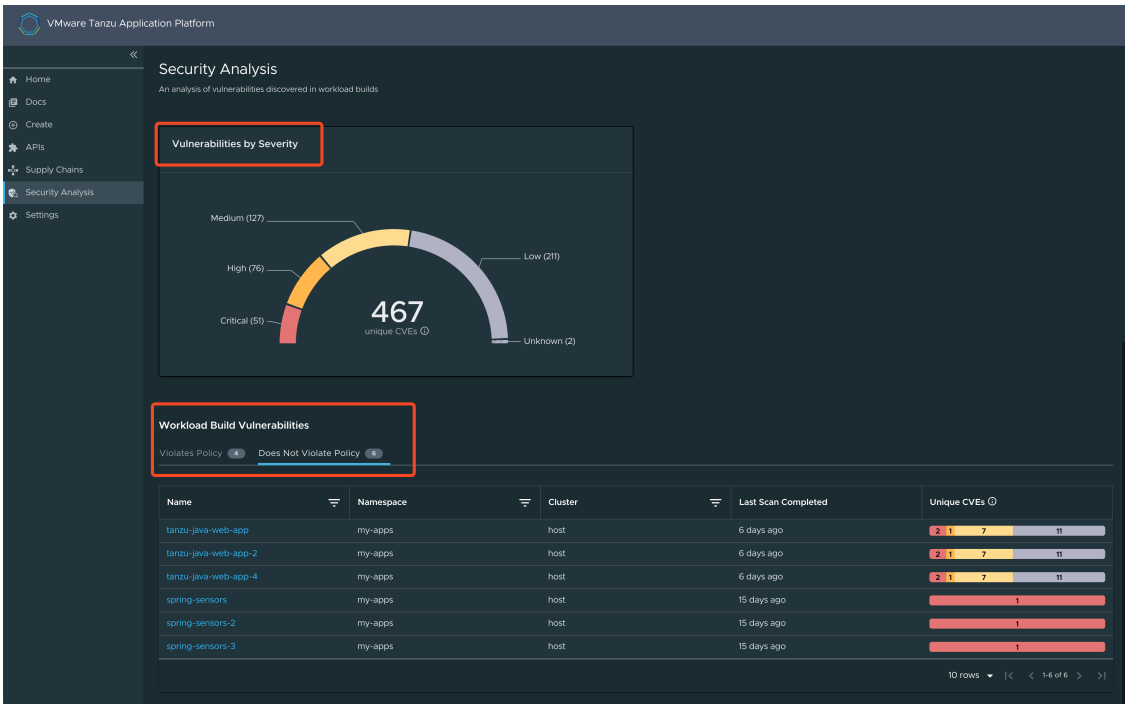
Accessing the plug-in

The Security Analysis plug-in is always accessible from the left navigation pane. Click the **Security Analysis** button to open the **Security Analysis** dashboard.



Viewing vulnerability data

The **Security Analysis** dashboard provides a summary of all vulnerabilities across all clusters for single-cluster and multicluster deployments.



The **Vulnerabilities by Severity** widget quickly counts the number of critical, high, medium, low, and unknown severity CVEs, based on the CVSS severity rating of each CVE.

It includes a sum of all workloads' source and image scan vulnerabilities. For example, if CVE-123 exists in the latest source scans and image scans of Workload ABC and Workload DEF, it is counted four times.

Note

The sum includes any CVEs on the allowlist (ignoreCVEs).

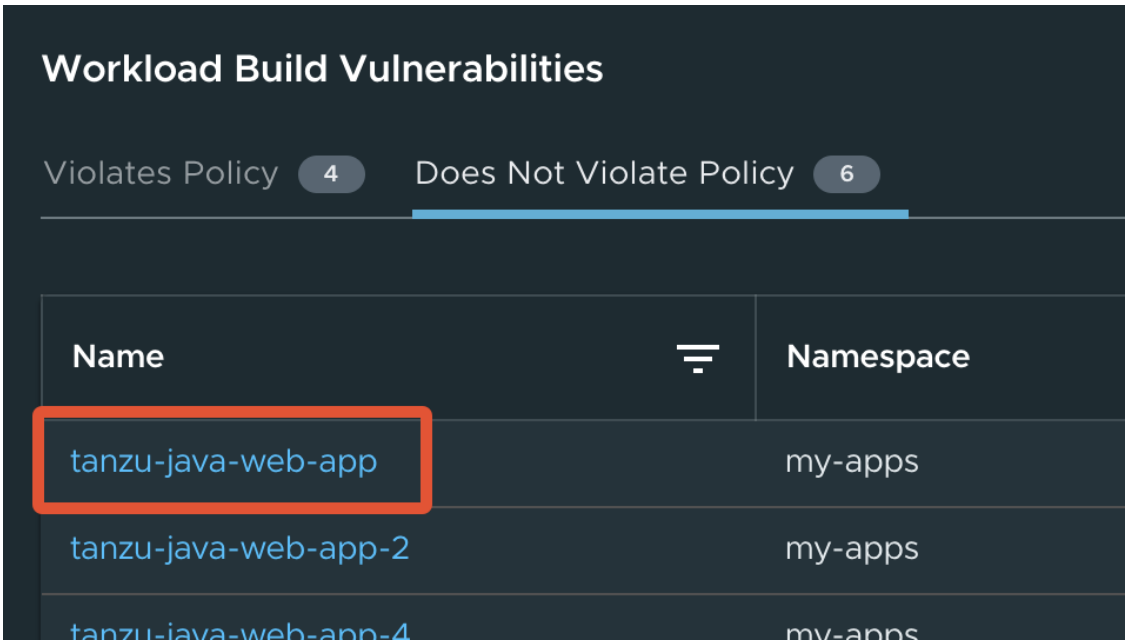
The **Workload Build Vulnerabilities** tables, with the **Violates Policy** tab and **Does Not Violate** tab, separate workloads based on the scan policy. For more information, see Enforce compliance policy using Open Policy Agent The Unique CVEs column uses the same sum logic as described earlier, but for individual workloads.

The sum of a workload's CVEs might not match the Supply Chain Choreographer's Vulnerability Scan Results. The data on this dashboard is based on `kubectl describe` for `SourceScan` and `ImageScan`. The data on the Supply Chain Choreographer's Vulnerability Scan Results is based on Metadata Store data.

Only vulnerability scans associated with a Cartographer workload appear. Use tanzu insight to view results for non-workload scan results.

Viewing CVE and package details

The Security Analysis plug-in has a **CVE** page and a **Package** page. These are accessed by clicking on a workload name, which opens the Supply Chain Choregrapher plug-in. Clicking on the CVE or Package name opens the **CVE** or **Package** page, respectively.



The **CVE** page contains basic information about the vulnerability and includes a table with all affected packages and versions.

The **Package** page contains basic information about a package and includes a table with all CVEs and the affected package versions.

Supply Chain Choreographer in Tanzu Application Platform GUI

This topic tells you about Supply Chain Choreographer in Tanzu Application Platform GUI (commonly called TAP GUI).

Overview

The Supply Chain Choreographer (SCC) plug-in enables you to visualize the execution of a workload by using any of the installed Out-of-the-Box supply chains. For more information about the Out-of-the-Box (OOTB) supply chains that are available in Tanzu Application Platform, see Supply Chain Choreographer for Tanzu.

Prerequisites

To use Supply Chain Choreographer in Tanzu Application Platform GUI you must have:

- One of the following installed on your cluster:
  - Tanzu Application Platform Full profile
  - Tanzu Application Platform View profile
  - Tanzu Application Platform GUI package and a metadata store package
- One of the following installed on the target cluster where you want to deploy your workload:
  - Tanzu Application Platform Run profile
  - Tanzu Application Platform Full profile

For more information, see Overview of multicluster Tanzu Application Platform

Enable CVE scan results

To see CVE scan results within Tanzu Application Platform GUI, connect Tanzu Application Platform GUI to the Tanzu Application Platform component Supply Chain Security Tools - Store (SCST - Store).

Automatically connect Tanzu Application Platform GUI to the Metadata Store

Tanzu Application Platform GUI has automation for enabling connection between Tanzu Application Platform GUI and SCST - Store. To use this automation, add it to the Tanzu Application Platform GUI section within your `tap-values.yaml` file.

The default value for `tap_gui.metadataStoreAutoconfiguration` is `false`. See the following example:

```
# tap-values.yaml

# ...
tap_gui:
  # ...
  metadataStoreAutoconfiguration: true
```

This file change creates a service account for the connection with privileges scoped only to the Metadata Store. In addition it mounts the token of the service account into the Tanzu Application Platform GUI pod and produces for you the `app_config` section necessary for Tanzu Application Platform GUI to communicate with SCST - Store.

Important

If your configuration includes a `/metadata-store` block, the automation doesn't overwrite the proxy block that you provide. To use the automation you must delete the block at `tap_gui.app_config.proxy["/metadata-store"]`.

Manually connect Tanzu Application Platform GUI to the Metadata Store

To manually enable CVE scan results:

1. Obtain the read-write token, which is created by default when installing Tanzu Application Platform. Alternatively, create an additional read-write service account.
2. Add this proxy configuration to the `tap-gui:` section of `tap-values.yaml`:

   ```
   tap_gui:
     app_config:
       proxy:
         /metadata-store:
           target: https://metadata-store-app.metadata-store:8443/api/v1
           changeOrigin: true
           secure: false
           headers:
             Authorization: "Bearer ACCESS-TOKEN"
             X-Custom-Source: project-star
   ```

   Where `ACCESS-TOKEN` is the token you obtained after creating a read-write service account.

Important

The `Authorization` value must start with the word `Bearer`.

Enable GitOps Pull Request Flow

Set up for GitOps and pull requests to enable the supply chain box-and-line diagram to show **Approve a Request** in the **Config Writer** stage details view when the **Config Writer** stage is clicked. For more information, see GitOps vs. RegistryOps.

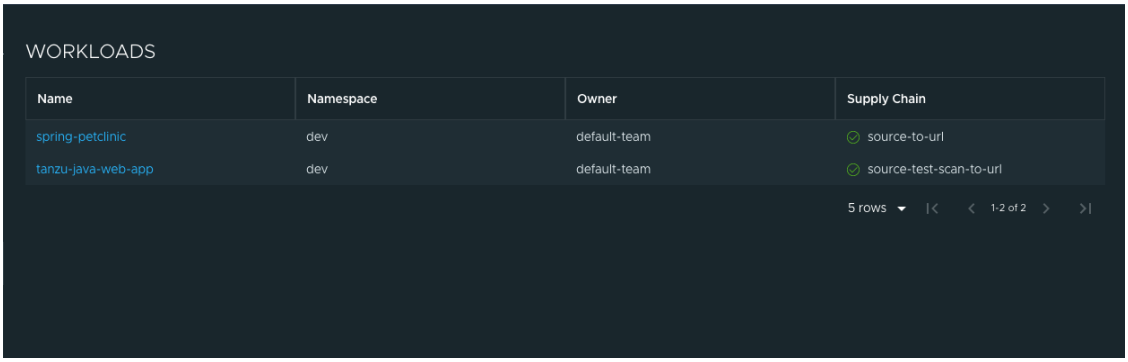
Supply Chain Visibility

Before using the Supply Chain Visibility (SCC) plug-in to visualize a workload, you must create a workload.

The workload must have the `app.kubernetes.io/part-of` label specified, whether you manually create the workload or use one supplied with the OOTB supply chains.

Use the left sidebar navigation to access your workload and visualize it in the supply chain that is installed on your cluster.

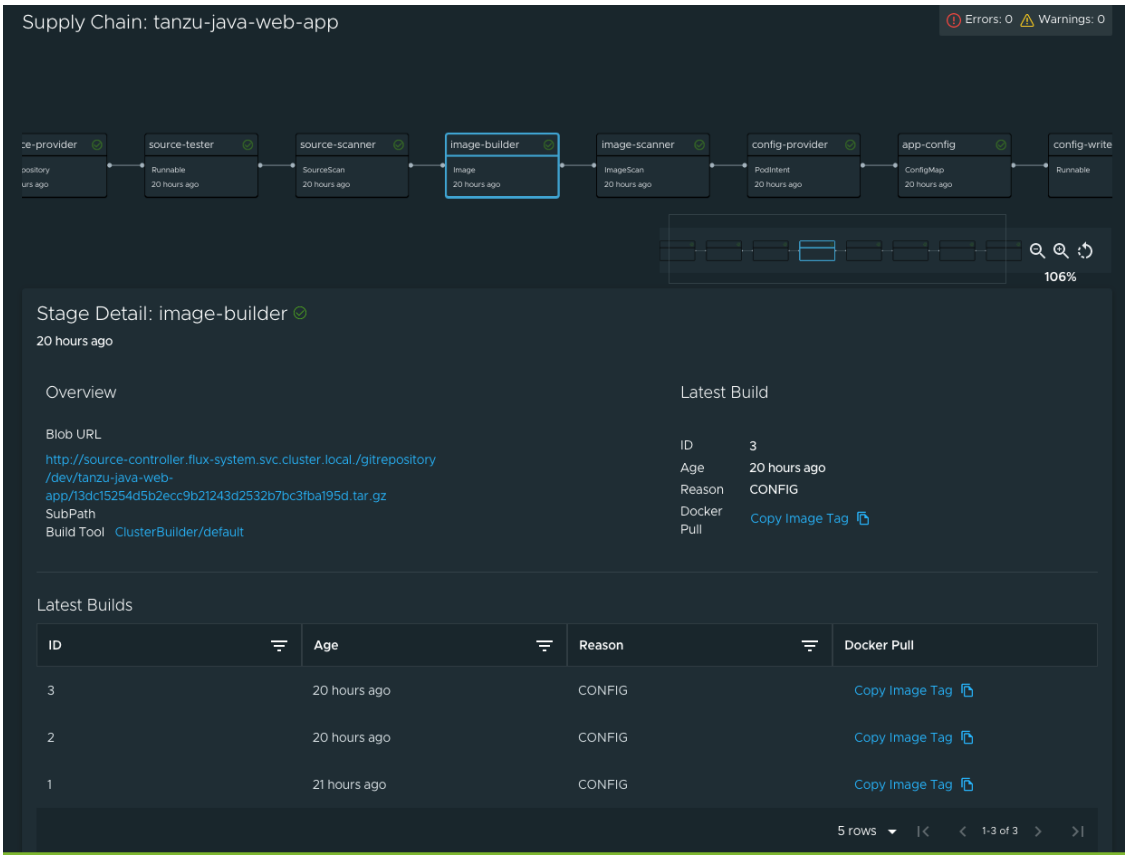
The example workload described in this topic is named `tanzu-java-web-app`.



Click **tanzu-java-web-app** in the **WORKLOADS** table to navigate to the visualization of the supply chain.

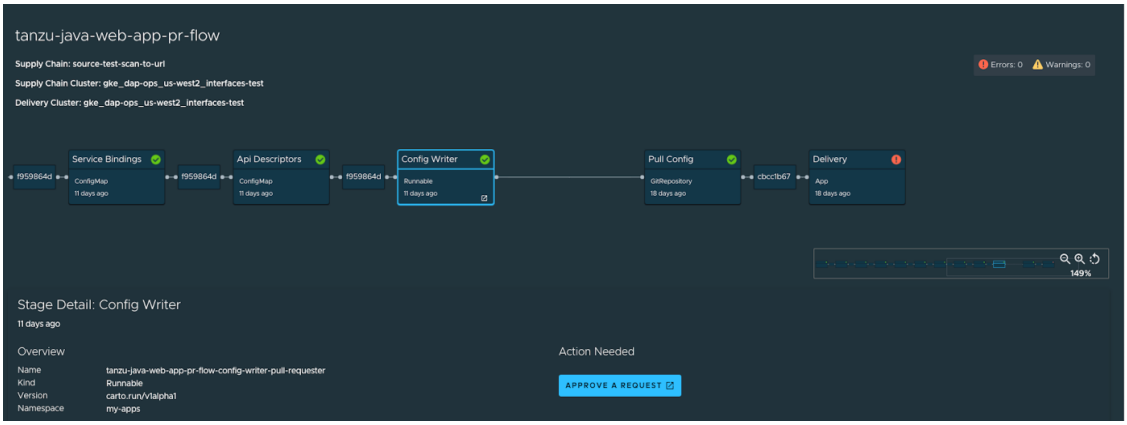
There are two sections within this view:

- The box-and-line diagram at the top shows all the configured CRDs that this supply chain uses, and any artifacts that the supply chain's execution outputs
- The **Stage Detail** section at the bottom shows source data for each part of the supply chain that you select in the diagram view



When a workload is deployed to a cluster that has the `deliverable` package installed, a new section appears in the supply chain that shows **Pull Config** boxes and **Delivery** boxes.

When you have a `Pull Request` configured in your environment, access the merge request from the supply chain by clicking **APPROVE A REQUEST**. This button is displayed after you click **Config Writer** in the supply chain diagram.



View Vulnerability Scan Results

Click the **Source Scan** stage or **Image Scan** stage to view vulnerability source scans and image scans for workload builds. The data is from Supply Chain Security Tools - Store.

CVE issues represent any vulnerabilities associated with a package or version found in the source code or image, including vulnerabilities from past scans.

Note

For example, the `log4shell` package is found in image ABC on 1 January without any CVEs. On 15 January, the log4j CVE issue is found while scanning image DEF. If a user returns to the **Image Scan** stage for image ABC, the log4j CVE issue appears and is associated with the `log4shell` package.

Overview of enabling TLS for Tanzu Application Platform GUI

Many users want inbound traffic to Tanzu Application Platform GUI (commonly called TAP GUI) to be properly encrypted. These topics tell you how to enable TLS encryption either with an existing certificate or by using the included cert-manager instance.

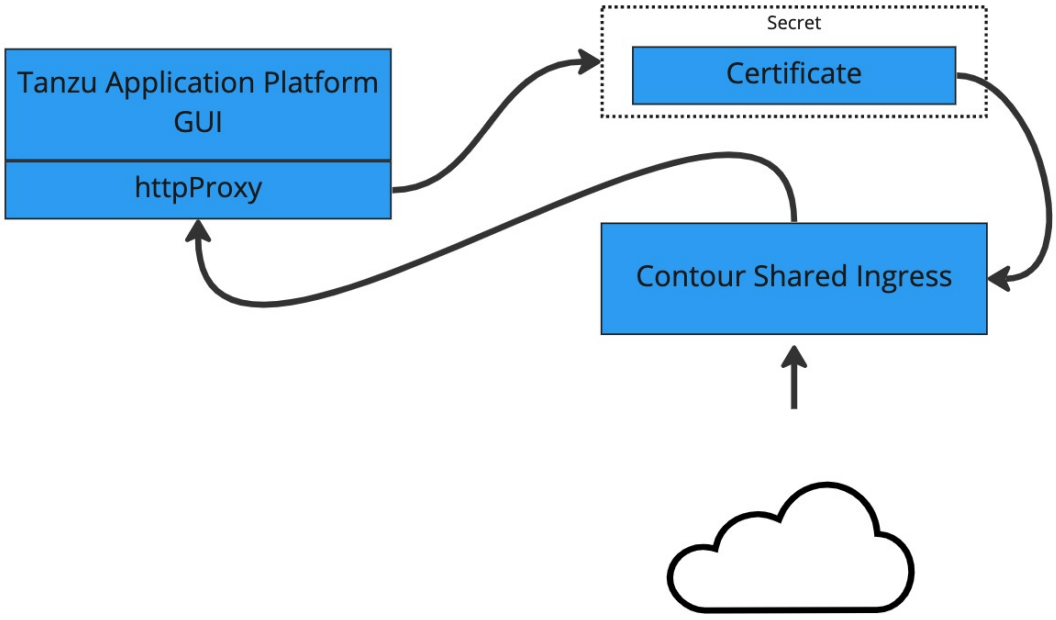
Concepts

The two key concepts are certificate delegation and the relationship between cert-manager, certificates, and ClusterIssuers.

Certificate delegation

Tanzu Application Platform GUI uses the established shared Contour ingress for TLS termination.

This enables you to store the certificate in a Kubernetes `secret` and then pass that `secret` and `namespace` to the `httpProxy` that was created during installation. To do this, see Configuring a TLS certificate by using an existing certificate.

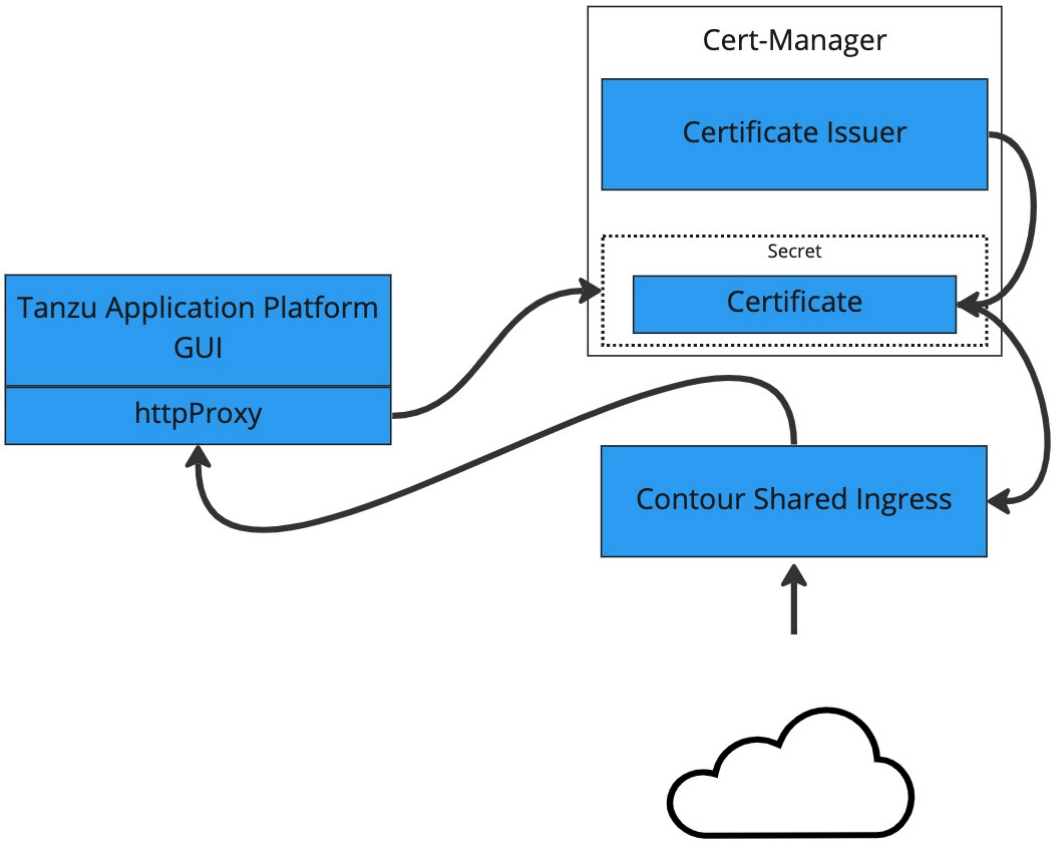


cert-manager, certificates, and ClusterIssuers

Tanzu Application Platform GUI can also use the `cert-manager` package that is installed when the profile was installed.

This tool allows cert-manager to automatically acquire a certificate from a `clusterIssuer` entity.

This external entity can be an external certificate authority, such as Let's Encrypt, or a self-signed certificate.



Guides

The following topics describe different ways to configure TLS:

- Configuring a TLS certificate by using an existing certificate
- Configuring a TLS certificate by using a self-signed certificate
- Configuring a TLS certificate by using cert-manager and a ClusterIssuer

Overview of enabling TLS for Tanzu Application Platform GUI

Many users want inbound traffic to Tanzu Application Platform GUI (commonly called TAP GUI) to be properly encrypted. These topics tell you how to enable TLS encryption either with an existing certificate or by using the included cert-manager instance.

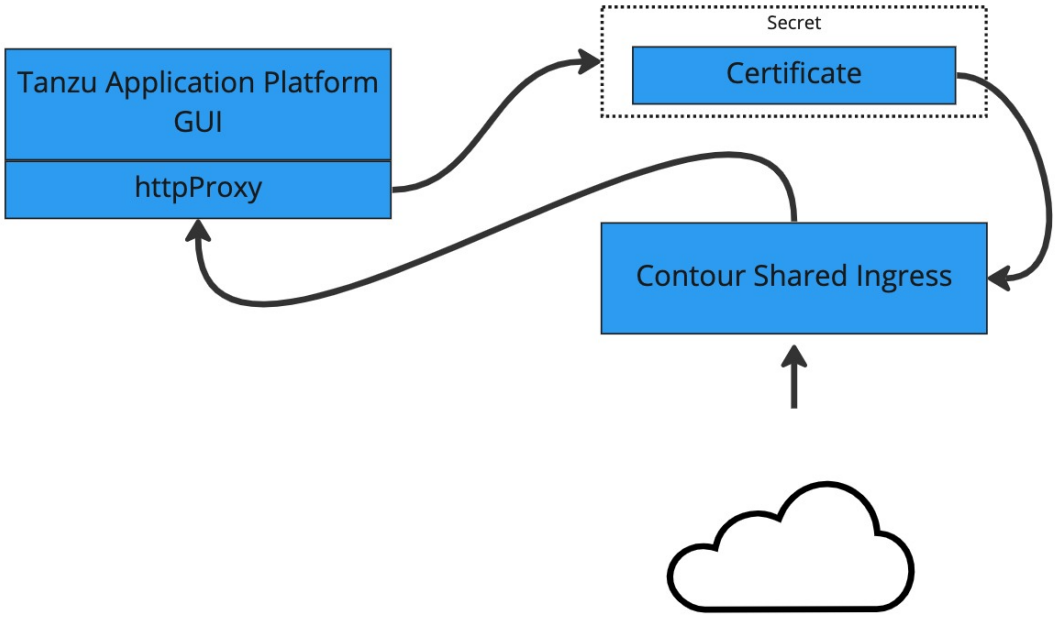
Concepts

The two key concepts are certificate delegation and the relationship between cert-manager, certificates, and ClusterIssuers.

Certificate delegation

Tanzu Application Platform GUI uses the established shared Contour ingress for TLS termination.

This enables you to store the certificate in a Kubernetes `secret` and then pass that `secret` and `namespace` to the `httpProxy` that was created during installation. To do this, see Configuring a TLS certificate by using an existing certificate.

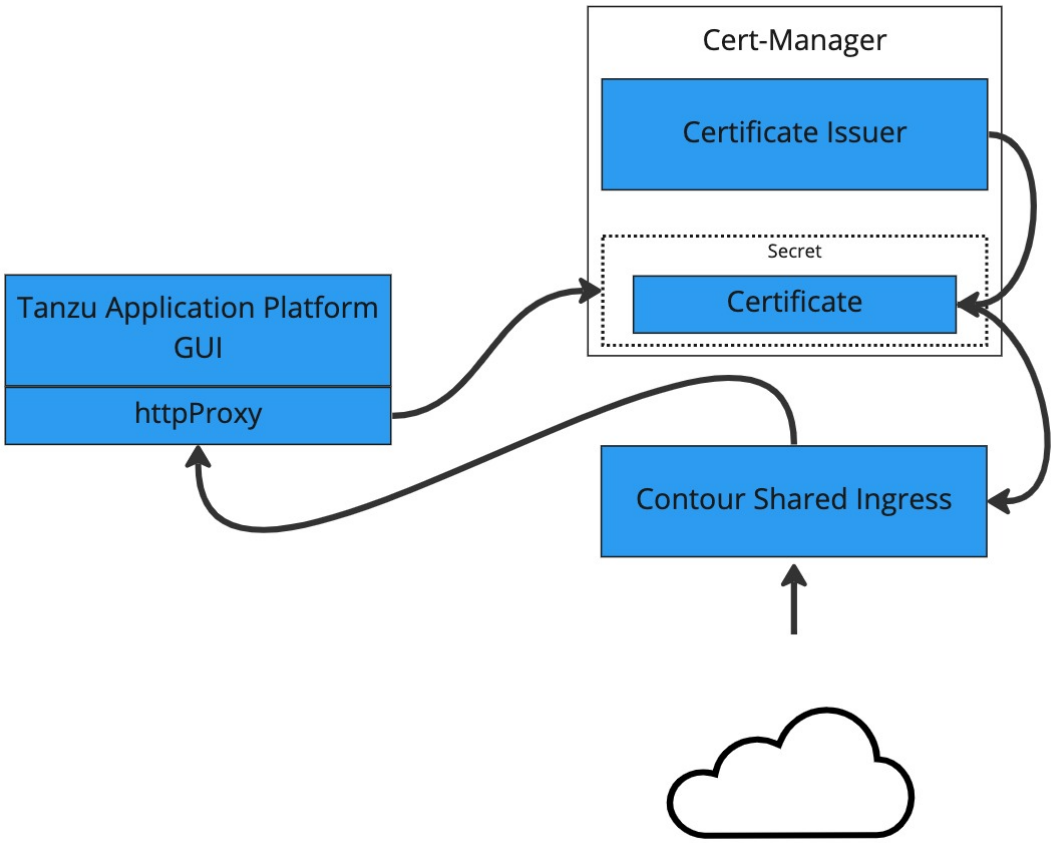


cert-manager, certificates, and ClusterIssuers

Tanzu Application Platform GUI can also use the `cert-manager` package that is installed when the profile was installed.

This tool allows cert-manager to automatically acquire a certificate from a `clusterIssuer` entity.

This external entity can be an external certificate authority, such as Let's Encrypt, or a self-signed certificate.



Guides

The following topics describe different ways to configure TLS:

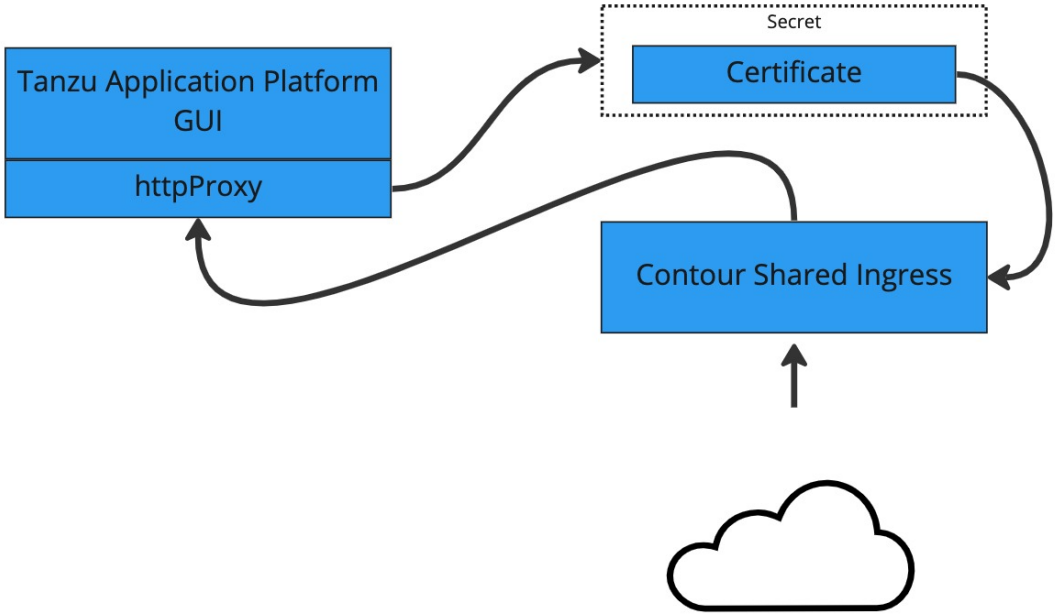
- Configuring a TLS certificate by using an existing certificate
- Configuring a TLS certificate by using a self-signed certificate
- Configuring a TLS certificate by using cert-manager and a ClusterIssuer

Configure a TLS certificate by using an existing certificate

This topic tells you how to use the certificate information from your external certificate authority to encrypt inbound traffic to Tanzu Application Platform GUI (commonly called TAP GUI).

Prerequisites

Your certificate authority gave you a certificate file, of the form `CERTIFICATE-FILE-NAME.crt`, and a signing key, of the form `KEY-FILE-NAME.key`. Ensure that these files are present on the host from which you run the CLI commands.



Procedure

To configure Tanzu Application Platform GUI with an existing certificate:

1. Create the Kubernetes secret by running:

   ```
   kubectl create secret tls tap-gui-cert --key="KEY-FILE-NAME.key" --cert="CERTIF
   ICATE-FILE-NAME.crt" -n tap-gui
   ```

   Where:

   - `KEY-FILE-NAME` is the name of the `key` file that your certificate issuer gave you
   - `CERTIFICATE-FILE-NAME` is the name of the `crt` file that your certificate issuer gave you

2. Configure Tanzu Application Platform GUI to use the newly created secret. Do so by editing the `tap-values.yaml` file that you used during installation to include the following under the `tap-gui` section:

   - A top-level `tls` key with subkeys for `namespace` and `secretName`
   - A namespace referring to the namespace used earlier
   - A secret name referring to the `secretName` value defined earlier

   Example:

   ```
   tap_gui:
    tls:
      namespace: tap-gui
      secretName: tap-gui-cert
   # Additional configuration below this line as needed
   ```

3. Update the Tanzu Application Platform package with the new values in `tap-values.yaml` by running:

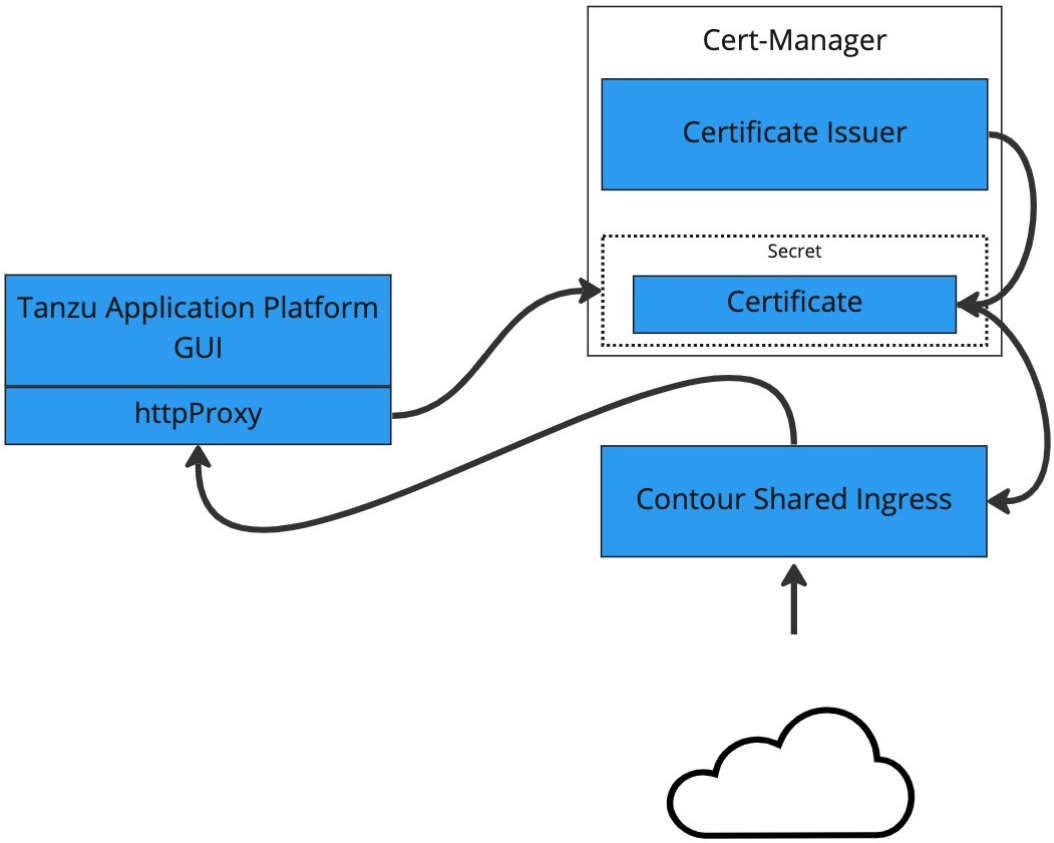
```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
lues-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version number that matches the values you used when you installed your profile.

Configure a TLS certificate by using a self-signed certificate

This topic tells you how to use cert-manager to create a self-signed certificate issuer and then generate a certificate for Tanzu Application Platform GUI to use based on that issuer.

Some browsers and corporate policies do not allow you to visit webpages that have self-signed certificates. You might need to navigate through a series of error messages to visit the page.



Prerequisite

Install a Tanzu Application Platform profile that includes cert-manager. Verify you did this by running the following command to detect the cert-manager namespace:

```
kubectl get ns
```

Procedure

To configure a self-signed TLS certificate for Tanzu Application Platform GUI:

1. Create a `certificate.yaml` file that defines an issuer and a certificate. For example:

```
apiVersion: cert-manager.io/v1
kind: Issuer
```

```
metadata:
 name: ca-issuer
 namespace: tap-gui
spec:
 selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
 name: tap-gui-cert
 namespace: tap-gui
spec:
 secretName: tap-gui-cert
 dnsNames:
 - tap-gui.INGRESS-DOMAIN
 issuerRef:
   name: ca-issuer
```

Where `INGRESS-DOMAIN` is your domain value that matches the values you used when you installed the profile.

2. Add the issuer and certificate to your cluster by running:

```
kubectl apply -f certificate.yaml
```

3. Configure Tanzu Application Platform GUI to use the newly created certificate. Update the `tap-values.yaml` file used during installation to include the following under the `tap-gui` section:

    - A top-level `tls` key with subkeys for `namespace` and `secretName`
    - A namespace referring to the namespace containing the `Certificate` object mentioned earlier
    - A secret name referring to the `secretName` value defined in your `Certificate` resource earlier

    Example:

```
tap_gui:
 tls:
   namespace: tap-gui
   secretName: tap-gui-cert
# Additional configuration below this line as needed
```

4. Update the Tanzu Application Platform package with the new values in `tap-values.yaml`:

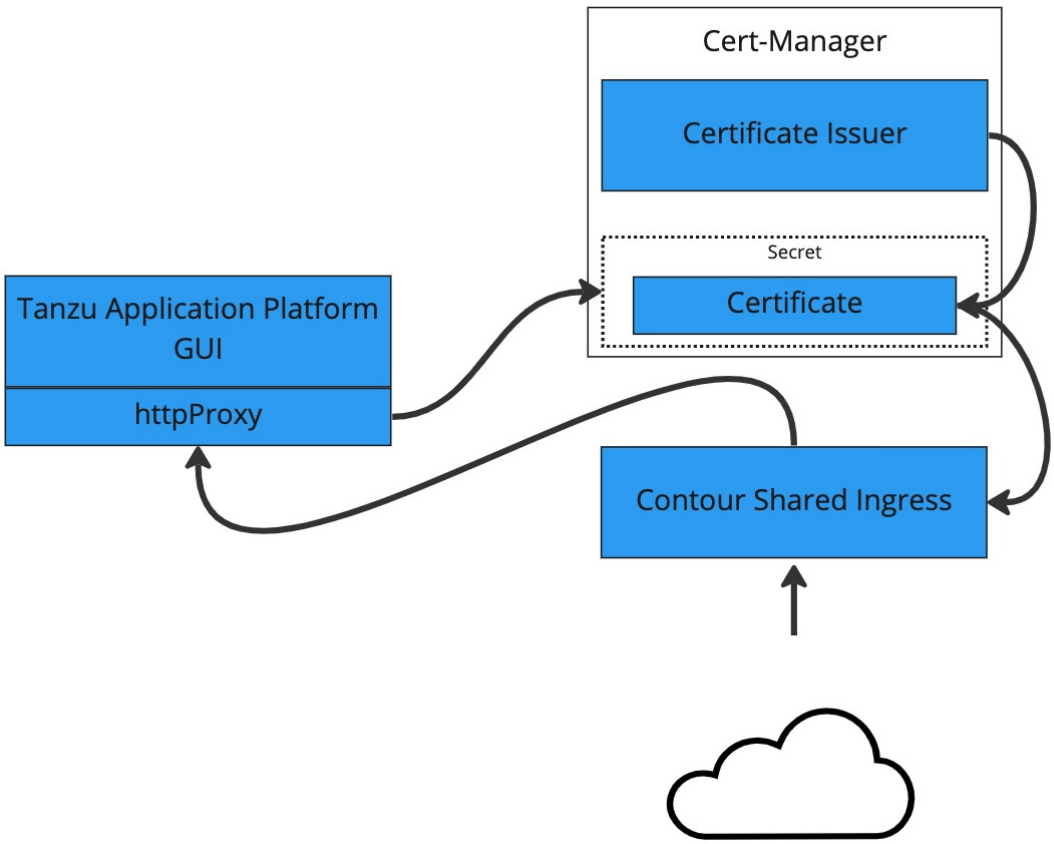
```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
lues-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version that matches the values you used when you installed the profile.

Configure a TLS certificate by using cert-manager and a ClusterIssuer

This topic tells you how to use cert-manager to create a certificate issuer and then generate a certificate for Tanzu Application Platform GUI (commonly called TAP GUI) to use based on that issuer.

This topic uses the free certificate issuer Let's Encrypt. You can use other certificate issuers compatible with cert-manager in a similar fashion.



Prerequisites

Fulfil these prerequisites:

- Install a Tanzu Application Platform profile that includes cert-manager. Verify you did this by running the following command to detect the cert-manager namespace:

  ```
  kubectl get ns
  ```

- Obtain a domain name that you control or own and have proof that you control or own it. In most cases, this domain name is the one you used for the `INGRESS-DOMAIN` values when you installed Tanzu Application Platform and Tanzu Application Platform GUI.
- If cert-manager cannot perform the challenge to verify your domain's compatibility, you must do so manually. For more information, see How It Works and Getting Started in the Let's Encrypt documentation.
- Ensure that your domain name is pointed at the shared Contour ingress for the installation. Find the IP address by running:

  ```
  kubectl -n tanzu-system-ingress get services envoy -o jsonpath='{.status.loadBa
  lancer.ingress[0].ip}'
  ```

Procedure

To configure a self-signed TLS certificate for Tanzu Application Platform GUI:

1. Create a `certificate.yaml` file that defines an issuer and a certificate. For example:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-http01-issuer
  namespace: cert-manager
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: EMAIL-ADDRESS
    privateKeySecretRef:
      name: letsencrypt-http01-issuer
    solvers:
    - http01:
        ingress:
          class: contour
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  namespace: cert-manager
  name: tap-gui
spec:
  commonName: tap-gui.INGRESS-DOMAIN
  dnsNames:
    - tap-gui.INGRESS-DOMAIN
  issuerRef:
    name: letsencrypt-http01-issuer
    kind: ClusterIssuer
  secretName: tap-gui
```

Where:

- `EMAIL-ADDRESS` is the email address that Let's Encrypt shows as responsible for this certificate
- `INGRESS-DOMAIN` is your domain value that matches the values you used when you installed the profile

2. Add the issuer and certificate to your cluster by running:

```
kubectl apply -f certificate.yaml
```

By applying the certificate, cert-manager attempts to perform an HTTP01 challenge by creating an Ingress resource specifically for the challenge. This is automatically removed from your cluster after the challenge is completed. For more information about how this works, and when it might not, see the cert-manager documentation.

3. Validate the certificate was created and is ready by running:

```
kubectl get certs -n cert-manager
```

Wait a few moments for this to take place, if need be.

4. Configure Tanzu Application Platform GUI to use the newly created certificate. To do so, update the `tap-values.yaml` file that you used during installation to include the following items under the `tap-gui` section:

- A top-level `tls` key with subkeys for `namespace` and `secretName`
- A namespace referring to the namespace containing the `Certificate` object from earlier
- A secret name referring to the `secretName` value defined in your `Certificate` resource earlier

Example:

```
tap_gui:
 tls:
   namespace: cert-manager
   secretName: tap-gui
# Additional configuration below this line as needed
```

5. Update the Tanzu Application Platform package with the new values in `tap-values.yaml` by running:

```
tanzu package installed update tap -p tap.tanzu.vmware.com -v TAP-VERSION  --va
lues-file tap-values.yaml -n tap-install
```

Where `TAP-VERSION` is the version that matches the values you used when you installed the profile.

Upgrade Tanzu Application Platform GUI

This topic tells you how to upgrade Tanzu Application Platform GUI (commonly called TAP GUI) outside of a Tanzu Application Platform profile installation. If you installed Tanzu Application Platform through a profile, see Upgrading Tanzu Application Platform instead.

Considerations

As part of the upgrade, Tanzu Application Platform updates its container with the new version.

As a result, if you installed Tanzu Application Platform GUI without the support of a backing database, you lose your in-memory data for any manual component registrations when the container restarts. While the update is pulling the new pod from the registry, users might experience a short UI interruption and might need to re-authenticate because the in-memory session data is rebuilt.

Upgrade within a Tanzu Application Platform profile

If you installed Tanzu Application Platform GUI as part of a Tanzu Application Platform profile, see Upgrading Tanzu Application Platform.

Upgrade Tanzu Application Platform GUI individually

These steps only apply to installing Tanzu Application Platform GUI individually, not as part of a Tanzu Application Platform profile.

To upgrade Tanzu Application Platform GUI outside of a Tanzu Application Platform profile:

1. Ensure that your repository has access to the new version of the package by running:

```
tanzu package available list tap-gui.tanzu.vmware.com -n tap-install
```

For example:

```
$ tanzu package available list tap-gui.tanzu.vmware.com -n tap-install
- Retrieving package versions for tap-gui.tanzu.vmware.com...
  NAME                      VERSION  RELEASED-AT
  tap-gui.tanzu.vmware.com  1.0.1    2021-12-22 17:45:51 +0000 UTC
  tap-gui.tanzu.vmware.com  1.0.2    2022-01-25 01:57:19 +0000 UTC
```

2. Perform the package upgrade by using the targeted package update version. Run:

```
tanzu package installed update tap-gui -p tap-gui.tanzu.vmware.com -v VERSION
--values-file \
TAP-GUI-VALUES.yaml -n tap-install
```

Where:

- `VERSION` is the target version of Tanzu Application Platform GUI that you want.
- `TAP-GUI-VALUES` is the configuration values file that contains the configuration used when you installed Tanzu Application Platform GUI.

3. Verify that you upgraded your application by running:

```
tanzu package installed get tap-gui -n tap-install
```

Troubleshoot Tanzu Application Platform GUI

This topic tells you how to troubleshoot issues encountered when installing Tanzu Application Platform GUI (commonly called TAP GUI). The topic is divided into sections:

- General issues
- Runtime Resources tab issues
- Accelerators page issues
- Security Analysis plug-in issues
- Supply Chain Choreographer plug-in issues

General issues

The following are general issues:

Tanzu Developer Portal reports that the port range is not valid

Symptom

You provided a full URL in a `backend.reading.allow` entry, as in this example `tap-values.yaml` snippet:

```
tap_gui:
  app_config:
    backend:
      reading:
        allow:
          - host: http://gitlab.example.com/some-group/some-repo/-/blob/main/catalog-i
nfo.yaml
```

and you see the following error message:

```
Backend failed to start up, Error: Port range is not valid: //gitlab.example.com/some-
group/some-repo/-/blob/main/catalog-info.yaml
```

Cause

Tanzu Application Platform GUI expects a host name to be passed into the field `backend.reading.allow[].host`.

Solution

Edit your `tap-values.yaml` file as in the following example:

```
tap_gui:
  app_config:
    backend:
      reading:
        allow:
          - host: gitlab.example.com
            paths: ['/some-group/some-repo/']
```

Tanzu Application Platform GUI does not load the catalog

Symptom

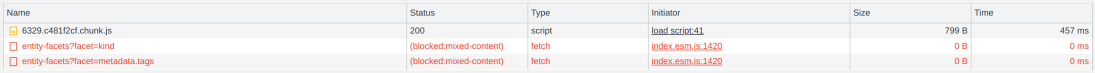
You are able to visit Tanzu Application Platform GUI, but it does not load the catalog and you see the following error message.

```
> Error: Could not fetch catalog entities.
> TypeError: Failed to fetch
```

When viewing your network tab you see that your browser has not downloaded mixed content. This might look different on different browsers.

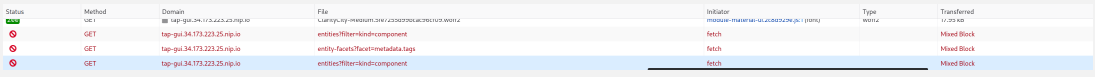
Chrome

In the **Status** column you see **(blocked:mixed-content)**



Firefox

In the **Transferred** column you see **Mixed Block**



Cause

As of Tanzu Application Platform v1.5, Tanzu Application Platform GUI provides TLS connections by default. Because of this, if you visit a Tanzu Application Platform GUI site your connection is automatically upgraded to https.

You might have manually set the fields `app.baseUrl`, `backend.baseUrl`, and `backend.cors.origin` in your `tap-values.yaml` file. Tanzu Application Platform GUI uses the `baseUrl` to determine how to create links to fetch from its APIs. The combination of these two factors causes your browser to attempt to fetch mixed content.

Solution

The solution is to delete these fields or update your values in `tap-values.yaml` to reflect that your Tanzu Application Platform GUI instance is serving https, as in the following example:

```
tap_gui:
  app_config:
    app:
      baseUrl: https://tap-gui.INGRESS-DOMAIN/
    backend:
      baseUrl: https://tap-gui.INGRESS-DOMAIN/
      cors:
        origin: https://tap-gui.INGRESS-DOMAIN/
```

Where `INGRESS-DOMAIN` is the ingress domain you have configured for Tanzu Application Platform.

The installer determines acceptable values based on your `tap_gui.ingressDomain` or `shared.ingress_domain` and the TLS status of the installation.

Updating a supply chain causes an error (`Can not create edge...`)

Symptom

Updating a supply chain causes an error (`Can not create edge...`) when an existing workload is clicked in the Workloads table and that supply chain is no longer present.

Solution

Recreate the same workload to execute through the new or updated supply chain.

Catalog not found

Symptom

When you pull up Tanzu Application Platform GUI, you get the error `Catalog Not Found`.

Cause

The catalog plug-in can't read the Git location of your catalog definition files.

Solution

1. Ensure you have built your own Backstage-compatible catalog or that you have downloaded one of the Tanzu Application Platform GUI catalogs from VMware Tanzu Network.
2. Ensure you defined the catalog in the values file that you input as part of installation. To update this location, change the definition file:
    - Change the Tanzu Application Platform profile file if installed by using a profile.
    - Change the standalone Tanzu Application Platform GUI values file if you're only installing that package on its own.

    ```
    namespace: tap-gui
    service_type: SERVICE-TYPE
    app_config:
      catalog:
        locations:
          - type: url
            target: https://GIT-CATALOG-URL/catalog-info.yaml
    ```

3. Provide the proper integration information for the Git location you specified earlier.

    ```
    namespace: tap-gui
    service_type: SERVICE-TYPE
    app_config:
      app:
        baseUrl: https://EXTERNAL-IP:PORT
      integrations:
        gitlab: # Other integrations available
          - host: GITLAB-HOST
    ```

```
            apiBaseUrl: https://GITLAB-URL/api/v4
            token: GITLAB-TOKEN
```

You can substitute for other integrations as defined in the Backstage documentation.

Issues updating the values file

Symptom

After updating the configuration of Tanzu Application Platform GUI, either by using a profile or as a standalone package installation, you don't know whether the configuration has reloaded.

Solution

1. Get the name you need by running:

   ```
   kubectl get pods -n tap-gui
   ```

   For example:

   ```
   $ kubectl get pods -n tap-gui
   NAME                      READY   STATUS    RESTARTS   AGE
   server-6b9ff657bd-hllq9   1/1     Running   0          13m
   ```

2. Read the log of the pod to see if the configuration reloaded by running:

   ```
   kubectl logs NAME -n tap-gui
   ```

   Where `NAME` is the value you recorded earlier, such as `server-6b9ff657bd-hllq9`.

3. Search for a line similar to this one:

   ```
   2021-10-29T15:08:49.725Z backstage info Reloaded config from app-config.yaml, a
   pp-config.yaml
   ```

4. If need be, delete and re-instantiate the pod.

   Caution

   Depending on your database configuration, deleting, and re-instantiating the pod might cause the loss of user preferences and manually registered entities. If you have configured an external PostgreSQL database, `tap-gui` pods are not stateful. In most cases, state is held in ConfigMaps, Secrets, or the database. For more information, see Configuring the Tanzu Application Platform GUI database and Register components.

   To delete and re-instantiate the pod, run:

   ```
   kubectl delete pod -l app=backstage -n tap-gui
   ```

Pull logs from Tanzu Application Platform GUI

Symptom

You have a problem with Tanzu Application Platform GUI, such as `Catalog: Not Found`, and don't have enough information to diagnose it.

Solution

Get timestamped logs from the running pod and review the logs:

1. Pull the logs by using the pod label by running:

   ```
   kubectl logs -l app=backstage -n tap-gui
   ```

2. Review the logs.

Ad-blocking software interference

Symptom

One or both of the following is true:

- You cannot access some, or all, of Tanzu Application Platform GUI.
- Telemetry collection within the VMware Customer Experience Improvement Program is failing.

Cause

Your ad-blocking browser extension or standalone ad-blocking software is causing interference.

Solution

Add Tanzu Application Platform GUI to your ad-blocking allowlist. Alternatively, deactivate the ad-blocking software or turn off Pendo telemetry collection.

TechDocs content does not load

Symptom

You navigate to the **Docs** page, click a document, and the content does not load. The loading bar does not disappear. The browser console shows the error message:

```
Refused to load the stylesheet 'https://fonts.googleapis.com/css?family=Roboto:300,400,400i,700%7CRoboto+Mono&display=fallback' because it violates the following Content Security Policy directive...
```

Cause

The Content Security Policy used by Tanzu Application Platform GUI is blocking `fonts.googleapis.com`.

Solution

Edit your `tap-values.yaml` file to include the CSP configuration, as in this example:

```
tap_gui:
  app_config:
    backend:
      csp:
        connect-src: ["'self'", 'http:', 'https:']
        img-src: ["'self'", 'https:', 'data:']
        style-src: ["'self'", "https:", "'unsafe-inline'"]
        upgrade-insecure-requests: false
```

Runtime Resources tab issues

Here are some common troubleshooting steps for errors presented in the **Runtime Resources** tab.

Error communicating with Tanzu Application Platform web server

Symptom

When accessing the **Runtime Resource Visibility** tab, the system displays `Error communicating with TAP GUI back end.`

Causes

- An interrupted Internet connection
- Error with the back end service

Solution

1. Confirm that you have Internet access.
2. Confirm that the back-end service is running correctly.
3. Confirm the cluster configuration is correct.

No data available

Symptom

When accessing the **Runtime Resource Visibility** tab, the system displays

```
One or more resources are missing. This could be due to a label mismatch. \
Please make sure your resources have the label(s) "LABEL_SELECTOR".
```

Cause

No communications error has occurred, but no resources were found.

Solution

Confirm that you are using the correct label:

1. Verify the Component definition includes the annotation `backstage.io/kubernetes-label-selector`.
2. Confirm your Kubernetes resources correspond to that label drop-down menu.

Errors retrieving resources

Symptom

When opening the **Runtime Resource Visibility** tab, the system displays `One or more resources might be missing because of cluster query errors.`

The reported errors might not indicate a real problem. A build cluster might not have runtime CRDs installed, such as Knative Service, and a run cluster might not have build CRDs installed, such as a Cartographer workload. In these cases, 403 and 404 errors might be false positives.

You might receive the following error messages because these resources might not be required for your specific Tanzu Application Platform profile. These error messages are known issues:

- `Access error when querying cluster CLUSTER_NAME for resource KUBERNETES_RESOURCE_PATH (status: 401). Contact your administrator.`
- `Access error when querying cluster CLUSTER_NAME for resource KUBERNETES_RESOURCE_PATH (status: 403). Contact your administrator.`
- `Knative is not installed on CLUSTER_NAME (status: 404). Contact your administrator.`
- `Error when querying cluster CLUSTER_NAME for resource KUBERNETES_RESOURCE_PATH (status: 404). Contact your administrator.`

Accelerators page issues

Here are some common troubleshooting steps for errors displayed on the **Accelerators** page.

No accelerators

Symptom

When the `app_config.backend.reading.allow` section is configured in the `tap-values.yaml` file during the `tap-gui` package installation, there are no accelerators on the **Accelerators** page.

Cause

This section in `tap-values.yaml` overrides the default configuration that gives Tanzu Application Platform GUI access to the accelerators.

Solution

As a workaround, provide a value for Application Accelerator in this section. For example:

```
app_config:
  # Existing tap-values yaml above
  backend:
    reading:
      allow:
      - host: acc-server.accelerator-system.svc.cluster.local
```

Security Analysis plug-in issues

These are troubleshooting issues for the Security Analysis plug-in.

Empty Impacted Workloads table

Symptom

The Impacted Workloads table is empty on the **CVE and Package Details** pages.

Cause

The relevant CVE belongs to a workload that has only completed one type of vulnerability scan (either image or source).

Solution

A fix is planned for Tanzu Application Platform GUI v1.5.1.

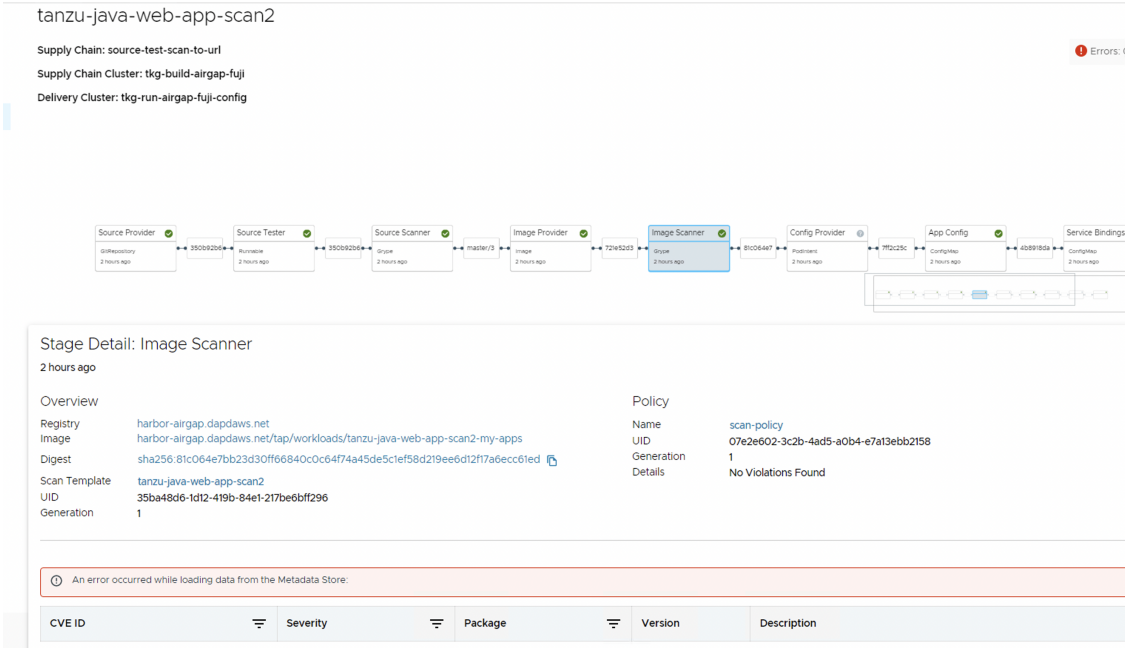
Supply Chain Choreographer plug-in issues

These are troubleshooting issues for the Supply Chain Choreographer plug-in.

An error occurred while loading data from the Metadata Store

Symptom

In the Supply Chain Choreographer plug-in, you see the error message `An error occurred while loading data from the Metadata Store`.



Cause

There are multiple potential causes. The most common cause is `tap-values.yaml` missing the configuration that enables Tanzu Application Platform GUI to communicate with Supply Chain Security Tools - Store.

Solution

See Supply Chain Choreographer - Enable CVE scan results for the necessary configuration to add to `tap-values.yaml`. After adding the configuration, update your Tanzu Application Platform deployment or Tanzu Application Platform GUI deployment with the new values.

Overview of Tanzu Application Platform Telemetry

Tanzu Application Platform Telemetry (commonly known as TAP Telemetry) is a set of objects that collect data about the usage of Tanzu Application Platform (commonly known as TAP) and send it back to VMware for product improvements.

A benefit of remaining enrolled in telemetry and identifying your company during Tanzu Application Platform installation is that VMware can provide your organization with usage reports about Tanzu Application Platform.

For more information about enrolling in telemetry reports, see Tanzu Application Platform usage reports.

For more information about how to install the telemetry component, see Install Tanzu Application Platform Telemetry.

Tanzu Application Platform usage reports

VMware offers the option to enroll in a usage reporting program that offers a summary of usage of your Tanzu Application Platform. You can enroll in the program by providing the Entitlement Account Number (EAN). An EAN is a unique ID assigned to all VMware customers. VMware uses EAN to identify data about Tanzu Application Platform. See Locate the Entitlement Account number for new orders for more details.

After locating the EAN, pass the number under the telemetry header in the `tap-values.yaml` file as a value for the `customer_entitlement_account_number` key.

```
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER"
```

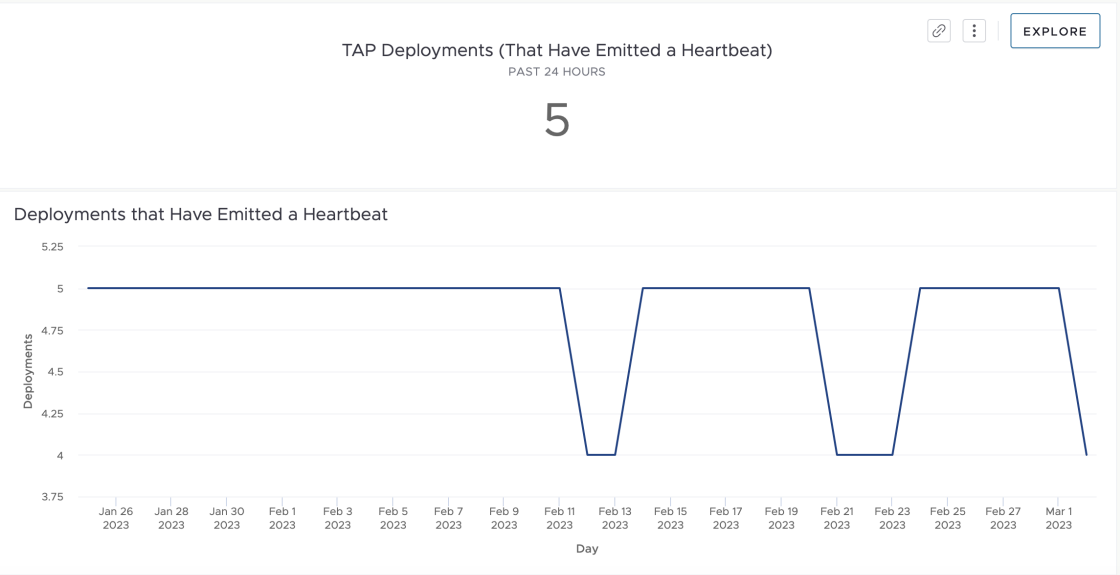
You must repeat the process for each Tanzu Application Platform Cluster included in the telemetry report. For more information, see Full profile.

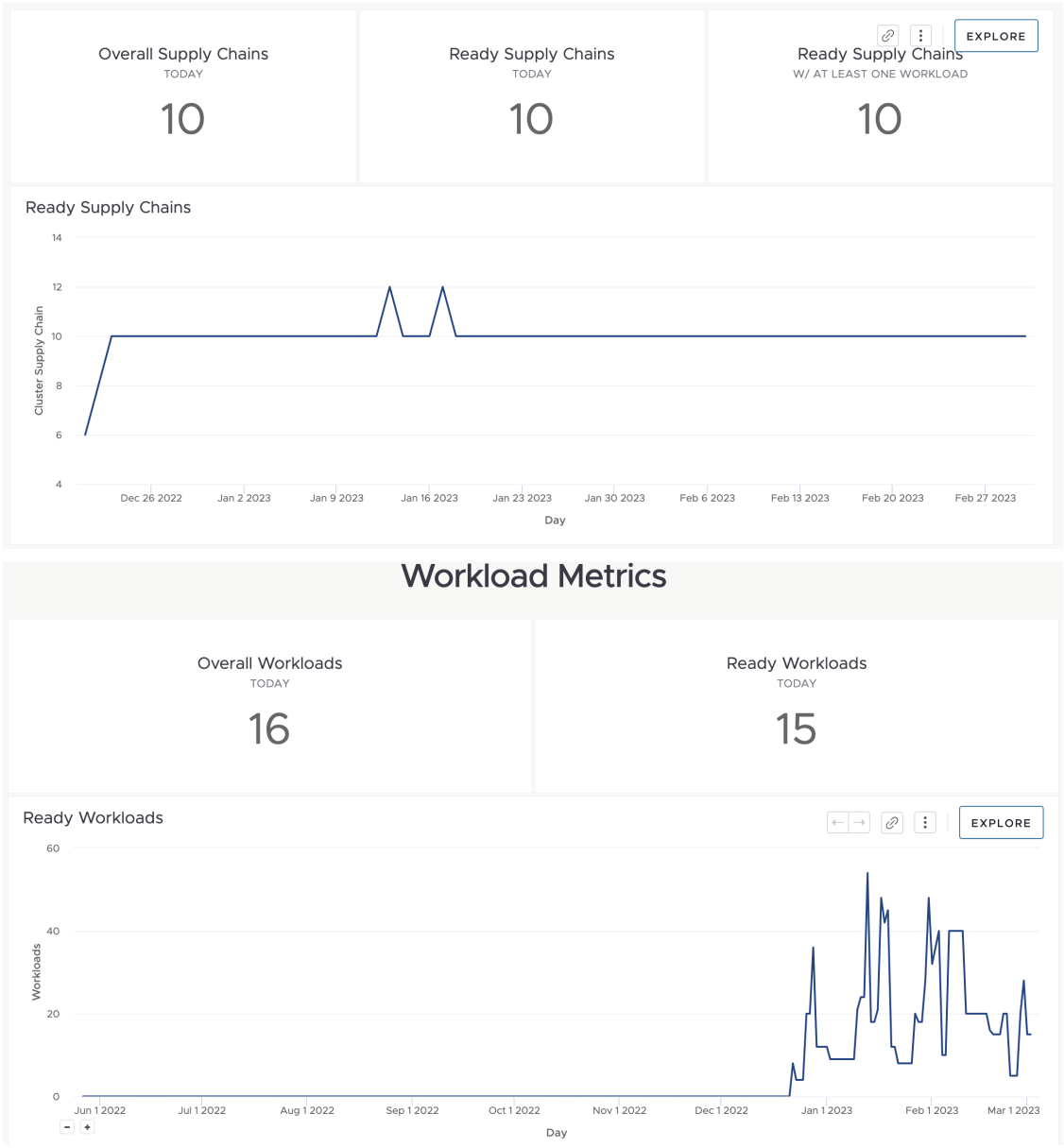
After enrollment, alert your VMware account team that you have configured the EAN field and want telemetry reports. This allows VMware to identify who the newly added EAN belongs to.

Note

Usage report is only supported for non-airgapped deployments of Tanzu Application Platform and the Cluster must participate in Tanzu Application Platform telemetry. You are enrolled in telemetry by default. You can opt out of telemetry collection by following the instructions in Opt out of telemetry collection.

The following screenshots show the sample telemetry reports.





Overview of Tanzu Application Platform Telemetry

Tanzu Application Platform Telemetry (commonly known as TAP Telemetry) is a set of objects that collect data about the usage of Tanzu Application Platform (commonly known as TAP) and send it back to VMware for product improvements.

A benefit of remaining enrolled in telemetry and identifying your company during Tanzu Application Platform installation is that VMware can provide your organization with usage reports about Tanzu Application Platform.

For more information about enrolling in telemetry reports, see Tanzu Application Platform usage reports.

For more information about how to install the telemetry component, see Install Tanzu Application Platform Telemetry.

Tanzu Application Platform usage reports

VMware offers the option to enroll in a usage reporting program that offers a summary of usage of your Tanzu Application Platform. You can enroll in the program by providing the Entitlement

Account Number (EAN). An EAN is a unique ID assigned to all VMware customers. VMware uses EAN to identify data about Tanzu Application Platform. See Locate the Entitlement Account number for new orders for more details.

After locating the EAN, pass the number under the telemetry header in the `tap-values.yaml` file as a value for the `customer_entitlement_account_number` key.

```
tap_telemetry:
  customer_entitlement_account_number: "CUSTOMER-ENTITLEMENT-ACCOUNT-NUMBER"
```

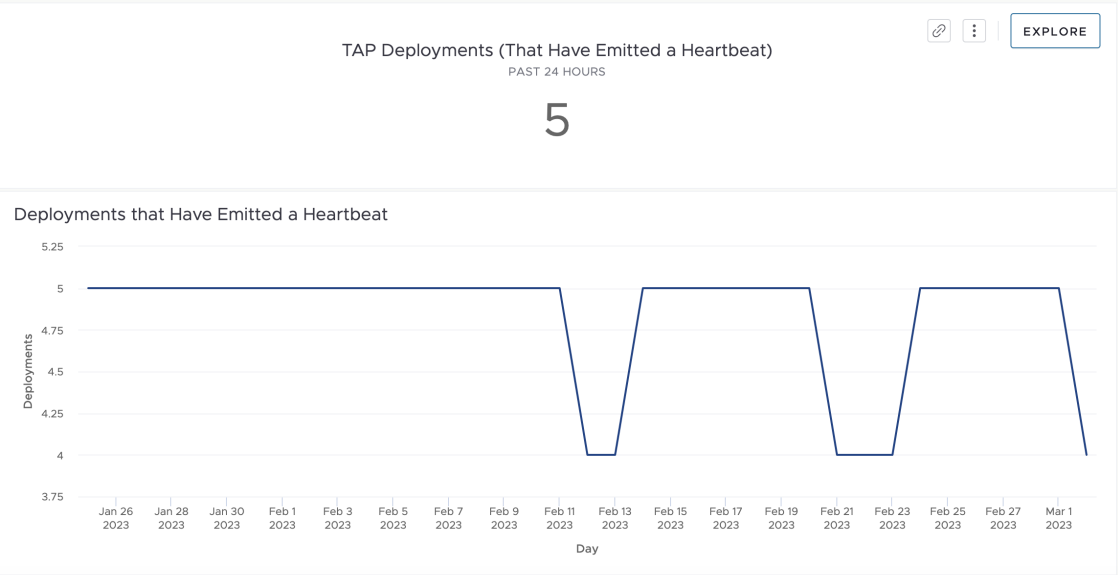
You must repeat the process for each Tanzu Application Platform Cluster included in the telemetry report. For more information, see Full profile.

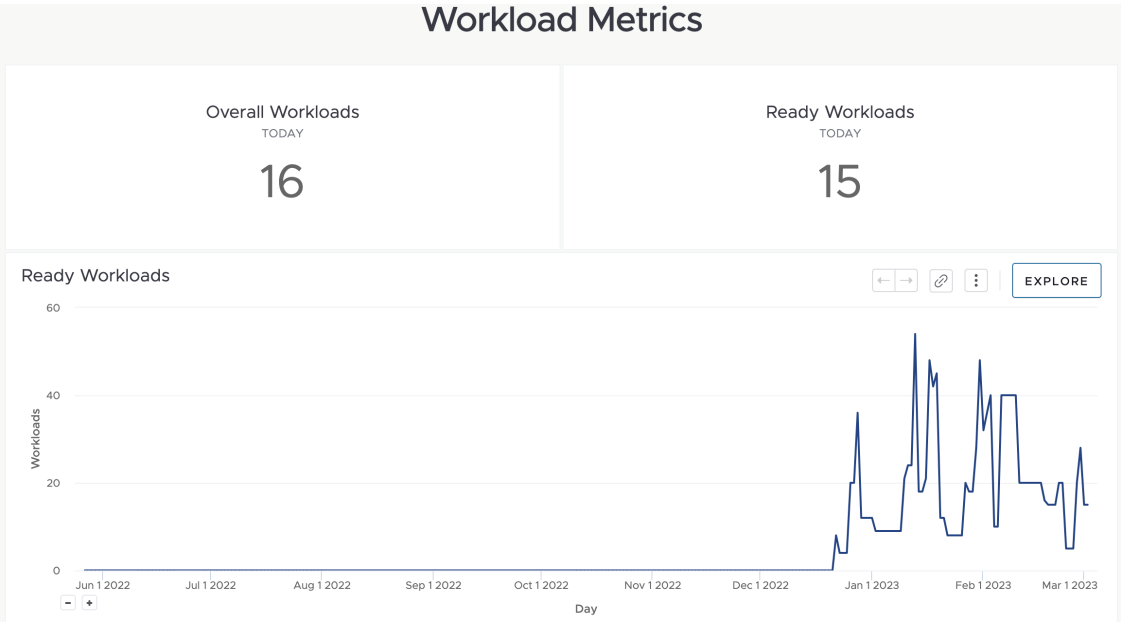
After enrollment, alert your VMware account team that you have configured the EAN field and want telemetry reports. This allows VMware to identify who the newly added EAN belongs to.

Note

Usage report is only supported for non-airgapped deployments of Tanzu Application Platform and the Cluster must participate in Tanzu Application Platform telemetry. You are enrolled in telemetry by default. You can opt out of telemetry collection by following the instructions in Opt out of telemetry collection.

The following screenshots show the sample telemetry reports.





Install Tanzu Application Platform Telemetry

This topic tells you how to install Tanzu Application Platform Telemetry from the Tanzu Application Platform (commonly known as TAP) package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Telemetry. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Tap Telemetry:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.

- Install cert-manager on the cluster. For more information, see the cert-manager documentation.
- See Deployment Details and Configuration to review what resources will be deployed.

Install

To install Tanzu Application Platform Telemetry:

1. List version information for the package by running:

   ```
   tanzu package available list tap-telemetry.tanzu.vmware.com --namespace tap-ins
   tall
   ```

   For example:

   ```
   $ tanzu package available list tap-telemetry.tanzu.vmware.com --namespace tap-i
   nstall
   - Retrieving package versions for tap-telemetry.tanzu.vmware.com...
     NAME                          VERSION       RELEASED-AT
     tap-telemetry.tanzu.vmware.com  0.3.1
   ```

2. (Optional) List all the available deployment configuration options:

   ```
   tanzu package available get tap-telemetry.tanzu.vmware.com/VERSION --values-sch
   ema -n tap-install
   ```

   Where `VERSION` is the your package version number. For example, `0.3.1`.

   For example:

   ```
   $ tanzu package available get tap-telemetry.tanzu.vmware.com/0.3.1 --values-sch
   ema -n tap-install
   | Retrieving package details for tap-telemetry.tanzu.vmware.com/0.3.1...
   KEY                                  DEFAULT  TYPE    DESCRIPTION
   kubernetes_distribution                       string  Kubernetes platform flavo
   r where the tap-telemetry is being installed on. Accepted values are ['', 'open
   shift']
   customer_entitlement_account_number           string  Account number used to di
   stinguish data by customer.
   installed_for_vmware_internal_use             string  Indication of if the depl
   oyment is for vmware internal user. Accepted values are ['true', 'false']
   ```

3. (Optional) Modify the deployment configurations by creating a configuration YAML with the desired custom configuration values. For example, if you want to provide your Customer Entitlement Number, create a `tap-telemetry-values.yaml` and configure the `customer_entitlement_account_number` property:

   ```
   ---
   customer_entitlement_account_number: "12345"
   ```

   See Deployment details and configuration for more information about the configuration options.

4. Install the package by running:

   ```
   tanzu package install tap-telmetry \
     --package tap-telemetry.tanzu.vmware.com \
     --version VERSION \
     --namespace tap-install \
     --values-file tap-telemetry-values.yaml
   ```

Where:

- `--values-file` is an optional flag. Only use it to customize the deployment configuration.
- `VERSION` is the package version number. For example, `0.3.1`.

For example:

```
$ tanzu package install tap-telmetry \
  --package tap-telemetry.tanzu.vmware.com \
  --version 0.3.1 \
  --namespace tap-install \
  --values-file tap-telemetry-values.yaml

  Installing package 'tap-telemetry.tanzu.vmware.com'
  Getting package metadata for 'tap-telemetry.tanzu.vmware.com'
  Creating service account 'tap-telemetry-tap-install-sa'
  Creating cluster admin role 'tap-telemetry-tap-install-cluster-role'
  Creating cluster role binding 'tap-telemetry-tap-install-cluster-rolebinding'
  Creating secret 'tap-telemetry-tap-install-values'
  Creating package resource
  Waiting for 'PackageInstall' reconciliation for 'tap-telemetry'
  'PackageInstall' resource install status: Reconciling
  'PackageInstall' resource install status: ReconcileSucceeded
  'PackageInstall' resource successfully reconciled

Added installed package 'tap-telemetry'
```

Deployment details and configurations of Tanzu Application Platform Telemetry

Use this topic to learn the deployment details and configurations of your Tanzu Application Platform Telemetry (commonly known as TAP Telemetry).

What is deployed

The installation creates the following in your Kubernetes cluster:

- A deployment.
- A pod.
- A namespace `tap-telemetry`.
- A service account with read-write privileges named `informer`, and a corresponding secret for the service account. This secret is bound to a ClusterRole named `tap-telemetry-admin`.
- A Role `tap-telemetry-informer` to retrieve the deployment ID, which is sent as sender ID in heartbeat metrics.
- A RoleBinding `tap-telemetry-informer-admin` that binds the `informer` service account to the `tap-telemetry-informer` role.
- A ClusterRole `tap-telemetry-admin` that has access to each Tanzu Application Platform component to gather information from.
- A ClusterRoleBinding `tap-telemetry-informer-admin` that binds the `informer` service account to the `tap-telemetry-informer` cluster role.

Deployment configuration

`customer_entitlement_account_number` is the unique identifier to differentiate between the data from your cluster and the data from other clusters. You can configure this property in your `tap-telemetry-values.yaml`:

```
customer_entitlement_account_number: "12345"
```

It creates a config map named `vmware-telemetry-identifiers` in the `vmware-system-telemetry` namespace, which is used internally to log your information.

Repeat these steps for the Build, Run, and View Cluster. For more information, see Install multicluster Tanzu Application Platform profiles.

Overview of Tanzu Build Service

This topic provides you with an overview of VMware Tanzu Build Service in Tanzu Application Platform (commonly known as TAP).

Overview

Tanzu Build Service automates container creation, management, and governance at enterprise scale. Tanzu Build Service uses the open-source Cloud Native Buildpacks project to turn application source code into container images. It executes reproducible builds aligned with modern container standards and keeps images up to date.

For more information about Tanzu Build Service, see the Tanzu Build Service documentation. For more information about Tanzu Buildpacks and their configuration, see the Tanzu Buildpack documentation.

Tanzu Application Platform v1.5 includes Tanzu Build Service v1.10.

Overview of Tanzu Build Service

This topic provides you with an overview of VMware Tanzu Build Service in Tanzu Application Platform (commonly known as TAP).

Overview

Tanzu Build Service automates container creation, management, and governance at enterprise scale. Tanzu Build Service uses the open-source Cloud Native Buildpacks project to turn application source code into container images. It executes reproducible builds aligned with modern container standards and keeps images up to date.

For more information about Tanzu Build Service, see the Tanzu Build Service documentation. For more information about Tanzu Buildpacks and their configuration, see the Tanzu Buildpack documentation.

Tanzu Application Platform v1.5 includes Tanzu Build Service v1.10.

Install Tanzu Build Service

This topic describes how to install Tanzu Build Service from the Tanzu Application Platform (commonly known as TAP) package repository by using the Tanzu CLI.

Before you begin

Use this topic if you do not want to use a Tanzu Application Platform profile that includes Tanzu Build Service. The Full, Iterate, and Build profiles include Tanzu Build Service. For more information about profiles, see Components and installation profiles.

The following procedure might not include some configurations required for your environment. For advanced information about installing Tanzu Build Service, see the Tanzu Build Service documentation.

Prerequisites

Before installing Tanzu Build Service:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- You must have access to a Docker registry that Tanzu Build Service can use to create builder images. Approximately 10 GB of registry space is required when using the `full` dependencies.
- Your Docker registry must be accessible with user name and password credentials.

Deprecated Features

- **The Cloud Native Buildpack Bill of Materials (CNB BOM) format:** For more information, see Deactivate the CNB BOM format.

Install the Tanzu Build Service package

To install Tanzu Build Service by using the Tanzu CLI:

1. Get the latest version of the Tanzu Build Service package by running:

   ```
   tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
   all
   ```

2. Gather the values schema by running:

   ```
   tanzu package available get buildservice.tanzu.vmware.com/VERSION --values-sche
   ma --namespace tap-install
   ```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.

3. Create the secret for the `kp-default-repository` credentials using the `tanzu` cli:

   ```
   tanzu secret registry add kp-default-repository-creds \
     --server "${REGISTRY_HOSTNAME}" \
     --username "${REGISTRY_USERNAME}" \
     --password "${REGISTRY_PASSWORD}" \
     --namespace tap-install
   ```

   Where: - `REGISTRY_HOST` is the hostname for the registry that will contain your `kp_default_repository`. Examples: - Harbor has the form `server: "my-harbor.io"`. - Docker Hub has the form `server: "index.docker.io"`. - Google Cloud Registry has the form `server: "gcr.io"`. - `REGISTRY_USERNAME` and `REGISTRY_PASSWORD` are the username and password for the user that can write to the repo used in the following step. For Google Cloud Registry, use `_json_key` as the username and the contents of the service account JSON file for the password.

4. Create a `tbs-values.yaml` file using the following template. If `shared.image_registry.project_path` and `shared.image_registry.secret` are configured in the `tap-values.yaml` file, Tanzu Build Service inherits all three values in that section. This can be disabled by setting any of the following three values.

```
---
kp_default_repository: "REPO-NAME"
kp_default_repository_secret:
  name: kp-default-repository-creds
  namespace: tap-install
```

   Where:

   `REPO-NAME` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location. Examples:

   - Harbor has the form `"my-harbor.io/my-project/build-service"`.
   - Docker Hub has the form `"my-dockerhub-user/build-service"` or `"index.docker.io/my-user/build-service"`.
   - Google Cloud Registry has the form `"gcr.io/my-project/build-service"`.

5. If you are running on Openshift, add `kubernetes_distribution: openshift` to your `tbs-values.yaml` file.

6. (Optional) Under the `ca_cert_data` key in the `tbs-values.yaml` file, provide a PEM-encoded CA certificate for Tanzu Build Service. This certificate is used for accessing the container image registry and is also provided to the build process.

   Note

   If `shared.ca_cert_data` is configured in the `tap-values.yaml` file, Tanzu Build Service inherits that value.

   Configuring `ca_cert_data` key in the `tbs-values.yaml` file adds the CA certificates at build time. To add CA certificates to the built image, see Configure custom CA certificates for a single workload using service bindings.

   For example:

```
---
kp_default_repository: "REPO-NAME"
kp_default_repository_secret:
  name: kp-default-repository-creds
  namespace: tap-install
ca_cert_data: |
  -----BEGIN CERTIFICATE-----
  ...
  -----END CERTIFICATE-----
```

7. (Optional) Tanzu Build Service is bootstrapped with the `lite` set of dependencies. To configure `full` dependencies, add the key-value pair `exclude_dependencies: true` to your `tbs-values.yaml` file. This is to exclude the default `lite` dependencies from the installation. For example:

```
---
kp_default_repository: "REPO-NAME"
kp_default_repository_secret:
```

```
  name: kp-default-repository-creds
  namespace: tap-install
exclude_dependencies: true
```

For more information about the differences between `full` and `lite` dependencies, see About lite and full dependencies.

8. Install the Tanzu Build Service package by running:

```
tanzu package install tbs \
  --package buildservice.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file tbs-values.yaml
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

For example:

```
$ tanzu package install tbs \
    --package buildservice.tanzu.vmware.com \
    --version 1.12.4 \
    --namespace tap-install \
    --vaules-file tbs-values.yaml

| Installing package 'buildservice.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'buildservice.tanzu.vmware.com'
| Creating service account 'tbs-tap-install-sa'
| Creating cluster admin role 'tbs-tap-install-cluster-role'
| Creating cluster role binding 'tbs-tap-install-cluster-rolebinding'
| Creating secret 'tbs-tap-install-values'
- Creating package resource
- Package install status: Reconciling

 Added installed package 'tbs' in namespace 'tap-install'
```

9. (Optional) Verify the cluster builders that the Tanzu Build Service installation created by running:

```
tanzu package installed get tbs -n tap-install
```

10. If you configured `full` dependencies in your `tbs-values.yaml` file, install the `full` dependencies by following the procedure in Install full dependencies.

Use AWS IAM authentication for registry credentials

Tanzu Build Service supports using AWS IAM roles to authenticate with Amazon Elastic Container Registry (ECR) on Amazon Elastic Kubernetes Service (EKS) clusters.

To use AWS IAM authentication:

1. Configure an AWS IAM role that has read and write access to the repository in the container image registry used when installing Tanzu Application Platform.
2. Use the following alternative configuration for `tbs-values.yaml`:

Note

> **Note**
>
> if you are installing Tanzu Build Service as part of a Tanzu Application Platform profile, you configure this in your `tap-values.yaml` file under the `buildservice` section.

```
---
  kp_default_repository: "REPO-NAME"
  kp_default_repository_aws_iam_role_arn: "IAM-ROLE-ARN"
```

Where:

- `REPO-NAME` is a writable repository in your registry. Tanzu Build Service dependencies are written to this location.
- `IAM-ROLE-ARN` is the AWS IAM role Amazon Resource Name (ARN) for the role configured in the previous step. For example, `arn:aws:iam::xyz:role/my-install-role`.

3. The developer namespace requires configuration for Tanzu Application Platform to use AWS IAM authentication for ECR. Configure an AWS IAM role that has read and write access to the registry for storing workload images.
4. Using the supply chain service account, add an annotation including the role ARN configured earlier by running:

```
kubectl annotate serviceaccount -n DEVELOPER-NAMESPACE SERVICE-ACCOUNT-NAME \
  eks.amazonaws.com/role-arn=IAM-ROLE-ARN
```

Where:

- `DEVELOPER-NAMESPACE` is the namespace where workloads are created.
- `SERVICE-ACCOUNT-NAME` is the supply chain service account. This is `default` if unset.
- `IAM-ROLE-ARN` is the AWS IAM role ARN for the role configured earlier. For example, `arn:aws:iam::xyz:role/my-developer-role`.

5. Apply this configuration by continuing the steps in Install the Tanzu Build Service package.

Install full dependencies

If you configured `full` dependencies in your `tbs-values.yaml` file, you must install the `full` dependencies package.

For a more information about `lite` and `full` dependencies, see About lite and full dependencies.

To install `full` Tanzu Build Service dependencies:

1. If you have not done so already, add the key-value pair `exclude_dependencies: true` to your `tbs-values.yaml` file. For example:

> **Note**
>
> if you are installing Tanzu Build Service as part of a Tanzu Application Platform profile, you configure this in your `tap-values.yaml` file under the `buildservice` section.

```
---
  kp_default_repository: "REPO-NAME"
  kp_default_repository_secret:
    name: kp-default-repository-creds
```

```
    namespace: tap-install
  exclude_dependencies: true
```

2. Get the latest version of the Tanzu Build Service package by running:

```
tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
all
```

3. Relocate the Tanzu Build Service `full` dependencies package repository by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
ps-package-repo:VERSION \
--to-repo INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps
```

   Where:

   - `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.
   - `INSTALL-REGISTRY-HOSTNAME` is your container image registry.
   - `TARGET-REPOSITORY` is your target repository.

4. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add tbs-full-deps-repository \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps:VERSION \
  --namespace tap-install
```

   Where:

   - `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.
   - `INSTALL-REGISTRY-HOSTNAME` is your container image registry.
   - `TARGET-REPOSITORY` is your target repository.

5. Install the `full` dependencies package by running:

```
tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
N -n tap-install
```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

(Optional) Deactivate the CNB BOM format

The legacy CNB BOM format is deprecated, but is enabled by default in Tanzu Application Platform.

To manually deactivate the format, add `include_legacy_bom=false` to either the `tbs-values.yaml` file, or to the `tap-values.yaml` file under the `buildservice` section.

Install Tanzu Build Service on an air-gapped environment

This topic tells you how to install Tanzu Build Service on a Kubernetes cluster and registry that are air-gapped from external traffic.

Before you begin

Use this topic if you do not want to use a Tanzu Application Platform profile that includes Tanzu Build Service.

The Full, Iterate, and Build profiles include Tanzu Build Service. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Tanzu Build Service:

- Complete all prerequisites to install Tanzu Application Platform. For more information, see Prerequisites.
- You must have access to a Docker registry that Tanzu Build Service can use to create builder images. Approximately 10 GB of registry space is required when using the `full` dependencies.
- Your Docker registry must be accessible with user name and password credentials.

Deprecated Features

The Cloud Native Buildpack Bill of Materials (CNB BOM) format: For more information, see Deactivate the CNB BOM format.

Install the Tanzu Build Service package

These steps assume that you have installed the Tanzu Application Platform packages in your air-gapped environment.

To install the Tanzu Build Service package on an air-gapped environment:

1. Get the latest version of the Tanzu Build Service package by running:

   ```
   tanzu package available list buildservice.tanzu.vmware.com --namespace tap-inst
   all
   ```

2. Gather the values schema by running:

   ```
   tanzu package available get buildservice.tanzu.vmware.com/VERSION --values-sche
   ma --namespace tap-install
   ```

   Where `VERSION` is the version of the Tanzu Build Service package you retrieved in the previous step.

3. Create a `tbs-values.yaml` file. The required fields for an air-gapped installation are as follows:

   ```
   ---
   kp_default_repository: REPO-NAME
   kp_default_repository_secret:
     name: "SECRET_NAME"
     namespace: "SECRET_NAMESPACE"
   ca_cert_data: CA-CERT-CONTENTS
   exclude_dependencies: true
   ```

   Where:

   - `REPO-NAME` is the fully qualified path to a writeable repository in your internal registry. Tanzu Build Service dependencies are written to this location. For example:
     - For Harbor: `harbor.io/my-project/build-service`
     - For Artifactory: `artifactory.com/my-project/build-service`

- `SECRET_NAME`/`SECRET_NAMESPACE` is the name/namespace of the secret containing credentials that can write to `REPO-NAME`.
- `CA-CERT-CONTENTS` are the contents of the PEM-encoded CA certificate for the internal registry.

4. Install the package by running:

```
tanzu package install tbs \
  --package buildservice.tanzu.vmware.com \
  --version VERSION \
  --namespace tap-install \
  --values-file tbs-values.yaml
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

For example:

```
$ tanzu package install tbs \
    --package buildservice.tanzu.vmware.com \
    --version 1.12.4 \
    --namespace tap-install \
    --values-file tbs-values.yaml

| Installing package 'buildservice.tanzu.vmware.com'
| Getting namespace 'tap-install'
| Getting package metadata for 'buildservice.tanzu.vmware.com'
| Creating service account 'tbs-tap-install-sa'
| Creating cluster admin role 'tbs-tap-install-cluster-role'
| Creating cluster role binding 'tbs-tap-install-cluster-rolebinding'
| Creating secret 'tbs-tap-install-values'
- Creating package resource
- Package install status: Reconciling
 Added installed package 'tbs' in namespace 'tap-install'
```

Install the Tanzu Build Service dependencies

By default, Tanzu Build Service is installed with `lite` dependencies.

When installing Tanzu Build Service on an air-gapped environment, the `lite` dependencies cannot be used as they require Internet access. You must install the `full` dependencies.

To install `full` dependencies:

1. Relocate the Tanzu Build Service `full` dependencies package repository by running:

```
imgpkg copy -b registry.tanzu.vmware.com/tanzu-application-platform/full-tbs-de
ps-package-repo:VERSION \
  --to-tar=tbs-full-deps.tar
# move tbs-full-deps.tar to environment with registry access
imgpkg copy --tar tbs-full-deps.tar \
  --to-repo=INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps
```

Where:

- `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.
- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.

2. Add the Tanzu Build Service `full` dependencies package repository by running:

```
tanzu package repository add tbs-full-deps-repository \
  --url INSTALL-REGISTRY-HOSTNAME/TARGET-REPOSITORY/tbs-full-deps:VERSION \
```

```
  --namespace tap-install
```

Where:

- `INSTALL-REGISTRY-HOSTNAME` is your container registry.
- `TARGET-REPOSITORY` is your target repository.
- `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

3. Install the `full` dependencies package by running:

```
tanzu package install full-tbs-deps -p full-tbs-deps.tanzu.vmware.com -v VERSIO
N -n tap-install
```

Where `VERSION` is the version of the Tanzu Build Service package you retrieved earlier.

Configure Tanzu Build Service properties on a workload

This topic tells you how to configure your workload with Tanzu Build Service properties.

Overview

Tanzu Build Service builds registry images from source code for Tanzu Application Platform. You can configure these build configurations by using a workload.

Tanzu Build Service is only applicable to the build process. Configurations, such as environment variables and service bindings, might require a different process for runtime.

Configure build-time service bindings

You can configure build-time service bindings for Tanzu Build Service.

Tanzu Build Service supports using the Service Binding Specification for Kubernetes for application builds. For more information, see the service binding specification for Kubernetes in GitHub.

Service binding configuration is specific to the buildpack that is used to build the app. For more information about configuring buildpack service bindings for the buildpack you are using, see the VMware Tanzu Buildpacks documentation.

To configure a service binding for a Tanzu Application Platform workload, follow these steps:

1. Create a YAML file named `service-binding-secret.yaml` for a secret as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: settings-xml
  namespace: DEVELOPER-NAMESPACE
type: service.binding/maven
stringData:
  type: maven
  provider: sample
  settings.xml: |
  MY-SETTINGS
```

Where: - `DEVELOPER-NAMESPACE` is the namespace where workloads are created. - `MY-SETTINGS` is the contents of your service bindings file.

2. Apply the YAML file by running:

```
kubectl apply -f service-binding-secret.yaml
```

3. Create the workload with `buildServiceBindings` configured by running:

   ```
   tanzu apps workload create WORKLOAD-NAME \
     --param-yaml buildServiceBindings='[{"name": "settings-xml", "kind": "Secre
   t"}]' \
     ...
   ```

   Where `WORKLOAD-NAME` is the name of the workload you want to configure.

Configure environment variables

If you have build-time environment variable dependencies, you can set environment variables that are available at build-time.

You can also configure buildpacks with environment variables. Buildpack configuration depends on the specific buildpack being used. For more information about configuring environment variables for the buildpack you are using, see the VMware Tanzu Buildpacks documentation.

For example:

```
tanzu apps workload create WORKLOAD-NAME \
  --build-env "ENV_NAME=ENV_VALUE" \
  --build-env "BP_MAVEN_BUILD_ARGUMENTS=-Dmaven.test.skip=true"
```

Where `WORKLOAD-NAME` is the name of the workload you want to configure.

Configure the service account

Using the Tanzu CLI, you can configure the service account used during builds. This service account is the one configured for the developer namespace. If unset, `default` is used.

To configure the service account used during builds, run:

```
tanzu apps workload create WORKLOAD-NAME \
  --param serviceAccount=SERVICE-ACCOUNT-NAME \
```

Where:

- `WORKLOAD-NAME` is the name of the workload you want to configure.
- `SERVICE-ACCOUNT-NAME` is the name of the service account you want to use during builds.

Configure the cluster builder

To configure the ClusterBuilder used during builds:

1. View the available ClusterBuilds by running:

   ```
   kubectl get clusterbuilder
   ```

2. Set the ClusterBuilder used during builds by running:

   ```
   tanzu apps workload create WORKLOAD-NAME \
     --param clusterBuilder=CLUSTER-BUILDER-NAME \
   ```

   Where:

   - `WORKLOAD-NAME` is the name of the workload you want to configure.
   - `CLUSTER-BUILDER-NAME` is the ClusterBuilder you want to use.

Configure the workload container image registry

Using the Tanzu CLI, you can configure the registry where workload images are saved. The service account used for this workload must have read and write access to this registry location.

To configure the registry where workload images are saved, run:

```
tanzu apps workload create WORKLOAD-NAME \
  --param-yaml registry={"server": SERVER-NAME, "repository": REPO-NAME}
```

Where:

- `SERVER-NAME` is the host name of the registry server. Examples:
  - Harbor has the form `"my-harbor.io"`.
  - Docker Hub has the form `"index.docker.io"`.
  - Google Cloud Registry has the form `"gcr.io"`.
- `REPO-NAME` is where workload images are stored in the registry. Images are written to `SERVER-NAME/REPO-NAME/workload-name`. Examples:
  - Harbor has the form `"my-project/supply-chain"`.
  - Docker Hub has the form `"my-dockerhub-user"`.
  - Google Cloud Registry has the form `"my-project/supply-chain"`.

Configure custom CA certificates for a single workload using service bindings

If the language family buildpack you are using includes the Paketo CA certificates buildpack, you can use a service binding to provide custom certificates during the build and run process. For more information about language family buildpacks, see the Tanzu Buildpacks documentation.

To create a service binding to provide custom CA certificates for a workload:

1. Create a YAML file named `service-binding-ca-cert.yaml` for a secret as follows:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: my-ca-certs
   data:
     type: ca-certificates
     provider: sample
     CA-CERT-FILENAME: |
       -----BEGIN CERTIFICATE-----
       ...
       -----END CERTIFICATE-----
   ```

   Where `CA-CERT-FILENAME` is the name of your PEM encoded CA certificate file. For example, `arbitrary-file-name.pem`.

2. Apply the YAML file by running:

   ```
   kubectl apply -f service-binding-ca-cert.yaml
   ```

3. To build with the custom certificate, create the workload with `--param-yaml buildServiceBindings` flag:

   ```
   tanzu apps workload create WORKLOAD-NAME \
     --param-yaml buildServiceBindings='[{"apiVersion": "v1", "kind": "Secret", "n
   ```

```
ame": "my-ca-certs"}]' \
  ...
```

Where `WORKLOAD-NAME` is the name of the workload you want to create.

4. To deploy with the custom certificate, create the workload with the `--service-ref` flag:

```
tanzu apps workload create WORKLOAD-NAME \
  --service-ref my-ca-certs=v1:Secret:my-ca-certs \
  ...
```

Where `WORKLOAD-NAME` is the name of the workload you want to create.

Using custom CA certificates for all workloads

To provide custom CA certificates to the build process for all workloads, see the optional step to add the `ca_cert_data` key Install the Tanzu Build Service package.

Create a signed container image with Tanzu Build Service

This topic tells you how to create a Tanzu Build Service image resource that builds a container image from source code signed with Cosign.

Prerequisites

Before you can configure Tanzu Build Service to sign your image builds, you must:

- Install Tanzu Build Service. The Full, Iterate, and Build profiles include Tanzu Build Service by default. If you have not installed Tanzu Application Platform with one of these profiles, see Installing Tanzu Build Service.
- Install Cosign. For instructions, see the Cosign documentation.
- Have a Builder or ClusterBuilder resource configured.
- Have an image resource configured.
- Review the kpack tutorial. This topic builds upon the steps in this tutorial.

Configure Tanzu Build Service to sign your image builds

To configure Tanzu Build Service to sign your image builds:

1. Ensure that you are in a Kubernetes context where you are authenticated and authorized to create and edit secret and service account resources.
2. Generate a Cosign keypair and store it as a Kubernetes secret by running:

```
cosign generate-key-pair k8s://NAMESPACE/COSIGN-KEYPAIR-NAME
```

Where:

- `NAMESPACE` is the namespace to store the Kubernetes secret in.
- `COSIGN-KEYPAIR-NAME` is the name of the Kubernetes secret.

For example:

```
cosign generate-key-pair k8s://default/tutorial-cosign-key-pair
```

3. Enter a password for the private key. Enter any password you want. After the command has completed, you will see the following output:

   ```
   Successfully created secret tutorial-cosign-key-pair in namespace default
   Public key written to cosign.pub
   ```

   You will also see a `cosign.pub` file in your current directory. Keep this file as you will need it to verify the signature of the images that are built.

4. If you are using Docker Hub or a registry that does not support OCI media types, add the annotation `kpack.io/cosign.docker-media-types: "1"` to the Cosign secret as follows:

   ```
   apiVersion: v1
   kind: Secret
   type: Opaque
   metadata:
     name: tutorial-cosign-key-pair
     namespace: default
     annotations:
       kpack.io/cosign.docker-media-types: "1"
   data:
     cosign.key: PRIVATE-KEY-DATA
     cosign.password: COSIGN-PASSWORD
     cosign.pub: PUBLIC-KEY-DATA
   ```

   For more information about configuring Cosign key pairs, see the Tanzu Build Service documentation.

5. To enable Cosign signing, create or edit the service account resource that is referenced in the image resource so that it includes the Cosign keypair secret created earlier. The service account is in the same namespace as the image resource and is directly referenced by the image or default if there isn't one. The default is the default service account in the workload namespace.

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: SERVICE-ACCOUNT-NAME
     namespace: default
   secrets:
   - name: REGISTRY-CREDENTIALS
   - name: COSIGN-KEYPAIR-NAME
   imagePullSecrets:
   - name: REGISTRY-CREDENTIALS
   ```

   Where:

   - `SERVICE-ACCOUNT-NAME` is the name of your service account resource. For example, `tutorial-cosign-service-account`.
   - `COSIGN-KEYPAIR-NAME` is the name of the Cosign keypair secret generated earlier. For example, `tutorial-cosign-key-pair`.
   - `REGISTRY-CREDENTIALS` is the secret that provides credentials for the container registry where application container images are pushed to.

6. Apply the service account resource to the cluster by running:

   ```
   kubectl apply -f cosign-service-account.yaml
   ```

7. Create an image resource file named `image-cosign.yaml`. For example:

```
apiVersion: kpack.io/v1alpha2
kind: Image
metadata:
  name: tutorial-cosign-image
  namespace: default
spec:
  tag: IMAGE-REGISTRY
  serviceAccountName: tutorial-cosign-service-account
  builder:
    name: my-builder
    kind: Builder
  source:
    git:
      url: https://github.com/spring-projects/spring-petclinic
      revision: 82cb521d636b282340378d80a6307a08e3d4a4c4
```

Where:

- `IMAGE-REGISTRY` with a writable repository in your registry. The secret referenced in the service account is a secret providing credentials for the registry where application container images are pushed to. For example:
  - Harbor has the form `"my-harbor.io/my-project/my-repo"`
  - Docker Hub has the form `"my-dockerhub-user/my-repo"` or `"index.docker.io/my-user/my-repo"`
  - Google Cloud Registry has the form `"gcr.io/my-project/my-repo"`

8. If you are using Out of the Box Supply Chains, edit the respective `ClusterImageTemplate` to enable signing in your supply chain. For more information, see Authoring supply chains.

Important

VMware discourages referencing the service account using the `service_account` value when installing the Out of the Box Supply Chain. This is because it gives your run cluster access to the private signing key.

9. Apply the image resource to the cluster by running:

```
kubectl apply -f image-cosign.yaml
```

10. After the image resource finishes building, you can get the fully resolved and built OCI image by running:

```
kubectl -n default get image tutorial-cosign-image
```

Example output:

```
NAME                  LATESTIMAGE                                          READY
tutorial-cosign-image index.docker.io/your-project/app@sha256:6744b...    True
```

11. Verify image signature by running:

```
cosign verify --insecure-ignore-tlog --key cosign.pub LATEST-IMAGE-WITH-DIGEST
```

Where `LATEST-IMAGE-WITH-DIGEST` is the value of `LATESTIMAGE` you retrieved in the previous step. For example, `index.docker.io/your-project/app@sha256:6744b...`.

Expected output:

```
Verification for index.docker.io/your-project/app@sha256:6744b... --
The following checks were performed on each of these signatures:
- The cosign claims were validated
- The signatures were verified against the specified public key
- Any certificates were verified against the Fulcio roots.
```

Note

You must use the `--insecure-ignore-tlog` flag because the supply chain does not write the signature attestation to a transparency log.

12. Configure Supply Chain Security Tools for VMware Tanzu - Policy Controller to ensure that only signed images are allowed in your cluster. For more information, see the Supply Chain Security Tools for VMware Tanzu - Policy Controller documentation.

Tanzu Build Service Dependencies

This topic tells you about Tanzu Build Service dependencies.

Important

Ubuntu Bionic will stop receiving support in April 2023. The Bionic stack is deprecated and will be removed in a future release. VMware recommends that you migrate builds to Jammy stacks. For Tanzu Application Platform v1.5 and later, the default stack for Tanzu Build Service is Jammy.

To build OCI images, Tanzu Build Service has the following dependencies: Cloud Native Buildpacks, Stacks, and Lifecycles.

How dependencies are installed

When Tanzu Application Platform is installed with Tanzu Build Service, it is bootstrapped with a set of dependencies. No extra configuration is required. Each version of Tanzu Application Platform and Tanzu Build Service contains new dependencies.

When Tanzu Application Platform is upgraded, new dependencies are installed which might cause workload images to rebuild. To ensure dependency compatibility, Tanzu Build Service only releases patches for dependencies in patch versions of Tanzu Application Platform. For upgrade instructions, see Upgrade the full dependencies package.

By default, Tanzu Build Service is installed with the `lite` set of dependencies, which are smaller-footprint and contain a subset of the buildpacks and stacks in the `full` set of dependencies. For a comparison of `lite` and `full` dependencies, see Dependency comparison later in this topic.

View installed dependencies

To view the set of dependencies installed with Tanzu Build Service, inspect the status of the cluster builders by running:

```
kubectl get clusterbuilder -o yaml
```

Cluster builders contain stack and buildpack metadata.

Bionic and Jammy stacks

Tanzu Application Platform v1.3 and later supports Ubuntu v22.04 (Jammy) based builds and will default to it from Tanzu Application Platform v1.5 and later.

Ubuntu Bionic will stop receiving support in April 2023. VMware recommends that you migrate builds to Jammy.

For more information about support for Jammy stacks, see About lite and full dependencies later in this topic.

Note

While upgrading apps to a newer stack, you might encounter the build platform erroneously reusing the old build cache. If you encounter this issue, delete and recreate the workload in Tanzu Application Platform, or delete and recreate the image in Tanzu Build Service.

About lite and full dependencies

Each version of Tanzu Application Platform is released with two types of Tanzu Build Service dependencies: `lite` and `full`. These dependencies consist of the buildpacks and stacks required for application builds. Each type serves different use cases. Both types are suitable for production workloads.

By default, Tanzu Build Service is installed with `lite` dependencies, which do not contain all buildpacks and stacks. To use all buildpacks and stacks, you must install the `full` dependencies. For instructions about installing `full` dependencies, see Install full dependencies.

For a table comparing the differences between `full` and `lite` dependencies, see Dependency comparison.

Lite dependencies

The `lite` dependencies are the default set installed with Tanzu Build Service.

`lite` dependencies contain a smaller footprint to speed up installation time, but do not support all workload types. For example, `lite` dependencies do not contain the PHP buildpack and cannot be used to build PHP workloads.

Lite dependencies: stacks

The `lite` dependencies contain the following stacks:

- `base` (Ubuntu Bionic)
- `base-jammy` (Ubuntu Jammy)
- `default` (identical to `base-jammy`)

For more information, see Stacks in the VMware Tanzu Buildpacks documentation.

Lite dependencies: buildpacks

The `lite` dependencies contain the following buildpacks in Tanzu Application Platform v1.5:

Buildpack	Version	Supported Stacks
Java Buildpack for VMware Tanzu (Lite)	8.8.0	Bionic, Jammy
Java Native Image Buildpack for Tanzu (Lite)	6.42.3	Bionic, Jammy

Buildpack	Version	Supported Stacks
.NET Core Buildpack for VMware Tanzu (Lite)	2.3.0	Bionic, Jammy
Node.js Buildpack for VMware Tanzu (Lite)	2.1.0	Bionic, Jammy
Python Buildpack for VMware Tanzu (Lite)	2.3.3	Bionic, Jammy
Go Buildpack for VMware Tanzu (Lite)	2.0.8	Bionic, Jammy
Web Servers Buildpack for VMware Tanzu (Lite)	0.8.0	Bionic, Jammy
Ruby Buildpack for VMware Tanzu (Lite)	2.1.0	Bionic, Jammy
Procfile Buildpack for VMware Tanzu (Lite)	5.4.0	Bionic, Jammy

And the following components:

Component	Version	Supported Stacks
CNB Lifecycle	0.16.0	Bionic, Jammy
Base Stack of Ubuntu Bionic for VMware Tanzu	1.2.45	Bionic
Base Stack of Ubuntu Jammy for VMware Tanzu	0.1.25	Jammy

Full dependencies

The Tanzu Build Service `full` set of dependencies contain more buildpacks and stacks, which allows for more workload types.

The dependencies are pre-packaged, so builds do not have to download them from the Internet. This can speed up build times and allows builds to occur in air-gapped environments. Due to the larger footprint of `full`, installations might take longer.

The `full` dependencies are not installed with Tanzu Build Service by default, you must install them. For instructions for installing `full` dependencies, see Install Tanzu Build Service with full dependencies.

Full dependencies: stacks

The `full` dependencies contain the following stacks, which support different use cases:

- `base` (Ubuntu Bionic)
- `full` (Ubuntu Bionic)
- `tiny` (Ubuntu Bionic)
- `base-jammy` (Ubuntu Jammy)
- `full-jammy` (Ubuntu Jammy)
- `tiny-jammy` (Ubuntu Jammy)
- `default` (identical to `base-jammy`)

For more information, see Stacks in the VMware Tanzu Buildpacks documentation.

Full dependencies: buildpacks

The `full` dependencies contain the following buildpacks in Tanzu Application Platform v1.5:

Buildpack	Version	Supported Stacks
Java Buildpack for VMware Tanzu	8.8.0	Bionic, Jammy

Buildpack	Version	Supported Stacks
Java Native Image Buildpack for Tanzu	6.42.3	Bionic, Jammy
.NET Core Buildpack for VMware Tanzu	2.3.0	Bionic, Jammy
Node.js Buildpack for VMware Tanzu	2.1.0	Bionic, Jammy
Python Buildpack for VMware Tanzu	2.3.3	Bionic, Jammy
Ruby Buildpack for VMware Tanzu	2.1.0	Bionic, Jammy
Go Buildpack for VMware Tanzu	2.0.8	Bionic, Jammy
PHP Buildpack for VMware Tanzu	2.0.0	Bionic, Jammy
Web Servers Buildpack for VMware Tanzu	0.8.0	Bionic, Jammy
Procfile Buildpack for VMware Tanzu	5.4.0	Bionic, Jammy

And the following components:

Component	Version	Supported Stacks
CNB Lifecycle	0.16.0	Bionic, Jammy
Tiny Stack of Ubuntu Bionic for VMware Tanzu	1.3.99	Bionic
Base Stack of Ubuntu Bionic for VMware Tanzu	1.2.45	Bionic
Full Stack of Ubuntu Bionic for VMware Tanzu	1.3.141	Bionic
Tiny Stack of Ubuntu Jammy for VMware Tanzu	0.1.26	Jammy
Base Stack of Ubuntu Jammy for VMware Tanzu	0.1.25	Jammy
Full Stack of Ubuntu Jammy for VMware Tanzu	0.1.54	Jammy

Dependency comparison

The following table compares the contents of the `lite` and `full` dependencies.

	lite	full
Faster installation time	Yes	No
Dependencies pre-packaged (faster builds)	No	Yes
Supports air-gapped installation	No	Yes
Contains base stack	Yes	Yes
Contains full stack	No	Yes
Contains tiny stack	No	Yes
Contains Jammy stack	Yes	Yes
Supports Java workloads	Yes	Yes
Supports Node.js workloads	Yes	Yes
Supports Go workloads	Yes	Yes
Supports Python workloads	Yes	Yes
Supports Ruby workloads	No	Yes
Supports .NET Core workloads	Yes	Yes

	lite	full
Supports PHP workloads	No	Yes
Supports static workloads	Yes	Yes
Supports binary workloads	Yes	Yes
Supports web servers buildpack	Yes	Yes

Updating dependencies

New versions of dependencies such as buildpacks, and stacks are available in new versions of Tanzu Application Platform. To update dependencies, VMware recommends that you update to the latest patch version of Tanzu Application Platform.

- If you are using `lite` or `full` dependencies, upgrade to the latest patch version of Tanzu Application Platform to update your dependencies.
- If you are using `full` dependencies, you must complete some extra steps after you upgrade to the latest patch. For more information, see Upgrading the full dependencies package.

Note

When Tanzu Application Platform is upgraded, new dependencies are installed which might cause workload images to rebuild.

Updating buildpack, and stack dependencies outside of upgrades to Tanzu Application Platform is possible but VMware does not recommend it, as we cannot guarantee those dependencies are compatible with the other components of Tanzu Application Platform. For more information about updating a stack and a buildpack, see Cluster stacks update and Cluster store update in the Tanzu Build Service documentation. Both workflows require the kp CLI.

Security context constraint for OpenShift

This topic tells you about running Tanzu Build Service on OpenShift clusters.

On OpenShift clusters Tanzu Build Service must run with a custom Security Context Constraint (SCC) to enable compliance. Tanzu Application Platform configures the following SCC for Tanzu Build Service when you configure the `kubernetes_distribution: openshift` key in the `tap-values.yaml` file.

```
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: tbs-restricted-scc-with-seccomp
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
  - NET_BIND_SERVICE
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
```

```
groups: []
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - ALL
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
seccompProfiles:
  - runtime/default
supplementalGroups:
  type: RunAsAny
users: []
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
```

It also applies the following RBAC to allow Tanzu Build Service services to use the SCC:

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    apps.tanzu.vmware.com/aggregate-to-workload: "true"
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  name: system:tbs:scc:restricted-with-seccomp
rules:
  - apiGroups:
      - security.openshift.io
    resourceNames:
      - tbs-restricted-scc-with-seccomp
    resources:
      - securitycontextconstraints
    verbs:
      - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:tbs:scc:restricted-with-seccomp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:tbs:scc:restricted-with-seccomp
subjects:
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: dependency-updater-controller-serviceaccount
  - kind: ServiceAccount
    namespace: build-service
    name: secret-syncer-service-account
  - kind: ServiceAccount
    namespace: build-service
    name: warmer-service-account
  - kind: ServiceAccount
    namespace: build-service
```

```
    name: build-service-daemonset-serviceaccount
  - kind: ServiceAccount
    namespace: cert-injection-webhook
    name: cert-injection-webhook-sa
  - kind: ServiceAccount
    namespace: kpack
    name: kp-default-repository-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: kpack-pull-lifecycle-serviceaccount
  - kind: ServiceAccount
    namespace: kpack
    name: controller
  - kind: ServiceAccount
    namespace: kpack
    name: webhook
  - kind: ServiceAccount
    namespace: stacks-operator-system
    name: controller-manager
```

Troubleshoot Tanzu Build Service

This topic tells you how to troubleshoot Tanzu Build Service when used with Tanzu Application Platform (commonly known as TAP).

Builds fail due to volume errors on EKS running Kubernetes v1.23

Symptom

After installing or upgrading Tanzu Application Platform on an Amazon Elastic Kubernetes Service (EKS) cluster running Kubernetes v1.23, build pods show:

```
'running PreBind plugin "VolumeBinding": binding volumes: timed out waiting
 for the condition'
```

Cause

This is due to the CSIMigrationAWS in this Kubernetes version, which requires users to install the Amazon EBS CSI driver to use AWS Elastic Block Store (EBS) volumes. For more information about EKS support for Kubernetes v1.23, see the Amazon blog post.

Tanzu Application Platform uses the default storage class which uses EBS volumes by default on EKS.

Solution

Follow the AWS documentation to install the Amazon EBS CSI driver before installing Tanzu Application Platform, or before upgrading to Kubernetes v1.23.

Smart-warmer-image-fetcher reports ErrImagePull due to dockerd's layer depth limitation

Symptom

When using dockerd as the cluster's container runtime, you might see the `smart-warmer-image-fetcher` pods report a status of `ErrImagePull`.

Cause

This error might be due to dockerd's layer depth limitation, in which the maximum supported image layer depth is 125.

To verify that the `ErrImagePull` status is due to dockerd's maximum supported image layer depth, check for event messages containing the words `max depth exceeded`. For example:

```
$ kubectl get events -A | grep "max depth exceeded"
  build-service        73s         Warning     Failed         pod/smart-warmer-image-f
etcher-wxtr8     Failed to pull image
  "harbor.somewhere.com/aws-repo/build-service:clusterbuilder-full@sha256:065bb361fd91
4a3970ad3dd93c603241e69cca214707feaa6
  d8617019e20b65e":  rpc error: code = Unknown desc = failed to register layer: max de
pth exceeded
```

Solution

To work around this issue, configure your cluster to use containerd or CRI-O as its default container runtime. For instructions, refer to the following documentation for your Kubernetes cluster provider.

For AWS, see:

- The Amazon blog
- The eksctl CLI documentation

For AKS, see:

- The Microsoft Azure documentation
- The Microsoft Azure blog

For GKE, see:

- The GKE documentation

For OpenShift, see:

- The Red Hat Hybrid Cloud blog
- The Red Hat Openshift documentation

Nodes fail due to "trying to send message larger than max" error

Symptom

You see the following error, or similar, in a node status:

```
Warning ContainerGCFailed 119s (x2523 over 42h) kubelet rpc error: code = ResourceExha
usted desc = grpc: trying to send message larger than max (16779959 vs. 16777216)
```

Cause

This is due to the way that the container runtime interface (CRI) handles garbage collection for unused images and containers.

Solution

Do not use Docker as the CRI because it is not supported. Some versions of EKS default to Docker as the runtime.

Build platform uses the old build cache after upgrade to new stack

Symptom

While upgrading apps to a newer stack, you might encounter the build platform erroneously reusing the old build cache.

Solution

If you encounter this issue, delete, and recreate the workload in Tanzu Application Platform, or delete and recreate the image in Tanzu Build Service.

Switching from `buildservice.kp_default_repository` to `shared.image_registry`

Symptom

After switching to using the `shared.image_registry` fields in `tap-values.yaml`, your workloads might start failing with a `TemplateRejectedByAPIServer` error, with the error message: `admission webhook "validation.webhook.kpack.io" denied the request: validation failed: Immutable field changed: spec.tag`.

Cause

Tanzu Application Platform automatically appends `/buildservice` to the end of the repository specified in `shared.image_registry.project_path`. This updates the existing workload image tags, which is not allowed by Tanzu Build Service.

Solution

Delete the `images.kpack.io`, it has the same name as the workload. The workload then recreates it with valid values.

Alternatively, re-add the `buildservice.kp_default_repository_*` fields in the `tap-values.yaml`. You must set both the repository and the authentication fields to override the shared values. Set `kp_default_repository`, `kp_default_repository_secret.name`, and `kp_default_repository_secret.namespace`.

Create a GitHub build action (Alpha)

This topic tells you how to use a GitHub action to create a Tanzu Build Service build on a cluster.

Important

Alpha features are experimental and are not ready for production use. Configuration and behavior is likely to change, and functionality might be removed in a future release.

Prerequisites

- Ensure that Tanzu Application Platform is installed.

Procedure

Developer namespace

1. Create a developer namespace where the build resource will be created.

   ```
   kubectl create namespace DEVELOPER-NAMESPACE
   ```

2. Create a service account in the `DEVELOPER-NAMESPACE` that has access to the registry credentials. This service account name will be used in the action.

Access to Kubernetes API server

The GitHub action talks directly to the Kubernetes API server, if you are running this on github.com with the default action runners, ensure that your API server is accessible from GitHub's IP ranges. Alternatively, it might be possible to run the action on a custom runner within your firewall (with access to the Tanzu Application Platform cluster).

Permissions Required

These are the minimum permissions required on the Tanzu Build Service cluster:

```
```bash
ClusterRole
 └ kpack.io
   └ clusterbuilders verbs=[get]
Role (DEVELOPER NAMESPACE)
 ├ ''
 │ ├ pods verbs=[get watch list] ✔
 │ └ pods/log verbs=[get] ✔
 └ kpack.io
   └ builds verbs=[get watch list create delete] ✔
```
```

This example contains the minimum required permissions:

```
```yaml
apiVersion: v1
kind: Namespace
metadata:
   name: DEVELOPER_NAMESPACE
---
apiVersion: v1
kind: ServiceAccount
metadata:
   namespace: DEVELOPER_NAMESPACE
   name: github-actions
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
   name: github-actions
subjects:
   - kind: ServiceAccount
     namespace: DEVELOPER_NAMESPACE
     name: github-actions
roleRef:
   kind: ClusterRole
   name: github-actions
```
```

```
   apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
   name: github-actions
   namespace: DEVELOPER_NAMESPACE
subjects:
   - kind: ServiceAccount
     namespace: DEVELOPER_NAMESPACE
     name: github-actions
roleRef:
   kind: Role
   name: github-actions
   apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
   name: github-actions
rules:
   - apiGroups: ['kpack.io']
     resources:
        - clusterbuilders
     verbs: ['get']
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
   name: github-actions
   namespace: DEVELOPER_NAMESPACE
rules:
   - apiGroups: ['']
     resources: ['pods']
     verbs: ['get', 'watch', 'list']
   - apiGroups: ['']
     resources: ['pods/log']
     verbs: ['get']
   - apiGroups: ['kpack.io']
     resources:
        - builds
     verbs: ['get', 'watch', 'list', 'create', 'delete']
```

To access the values on Google Kubernetes Engine (steps might vary on other IaaS providers):

```
  ```console
  DEV_NAMESPACE=DEVELOPER_NAMESPACE
  SECRET=$(kubectl get sa github-actions -oyaml -n $DEV_NAMESPACE | yq '.secrets[0].na
me')

  CA_CERT=$(kubectl get secret $SECRET -oyaml -n $DEV_NAMESPACE | yq '.data."ca.crt"')
  NAMESPACE=$(kubectl get secret $SECRET -oyaml -n $DEV_NAMESPACE | yq .data.namespace
| base64 -d)
  TOKEN=$(kubectl get secret $SECRET -oyaml -n $DEV_NAMESPACE | yq .data.token | base6
4 -d)
  SERVER=$(kubectl config view --minify | yq '.clusters[0].cluster.server')
  ```
```

Create the required secrets on the repository through GitHub.com or through the `gh` CLI:

```
```bash
gh secret set CA_CERT --app actions --body "$CA_CERT"
gh secret set NAMESPACE --app actions --body "$NAMESPACE"
gh secret set TOKEN --app actions --body "$TOKEN"
```
```

```
gh secret set SERVER --app actions --body "$SERVER"
```

Use the action

1. To use the action in a workflow, run the following YAML:

   ```
   - uses: vmware-tanzu/build-image-action@v1-alpha
     with:
       ## Authorization
       # Host of the API server
       server: `${{ secrets.SERVER }}`
       # CA Certificate of the API Server
       ca_cert: `${{ secrets.CA_CERT }}`
       # Service Account token to access Kubernetes
       token: `${{ secrets.TOKEN }}`
       # _(required)_ The namespace to create the build resource in
       namespace: `${{ secrets.NAMESPACE }}`

       ## Image configuration
       # _(required)_ Destination for the built image
       # Example: gcr.io/<my-project>/<my-image>
       destination: ''
       # Optional list of build time environment variables
       env: ''
       # Name of the service account in the namespace, defaults to `default`
       serviceAccountName: ''
       # Name of the cluster builder to use, defaults to `default`
       clusterBuilder: ''
       # Max active time that the pod can run for in seconds, defaults to 3600
       timeout:
   ```

   For example:

   ```
   - name: Build Image
     id: build
     uses: vmware-tanzu/build-image-action@v1-alpha
     with:
       # Authorization
       server: ${{ secrets.SERVER }}
       token: ${{ secrets.TOKEN }}
       ca_cert: ${{ secrets.CA_CERT }}
       namespace: ${{ secrets.NAMESPACE }}
       # Image configuration
       destination: gcr.io/project-id/name-for-image
       serviceAccountName: my-sa-that-has-access-to-reg-credentials
       env: |
         BP_JAVA_VERSION=17
   ```

2. The previous step should output the full name, including the SHA of the built image. To use the output in a subsequent step:

   ```
   - name: Do something with image
     run:
       echo "${{ steps.build.outputs.name }}"
   ```

Debugging

To run this action in “debug” mode, add a secret called `ACTIONS_STEP_DEBUG` with the value set to `true` as documented in the GitHub Action Docs.

Overview of Tanzu Developer Tools for IntelliJ

Tanzu Developer Tools for IntelliJ is the official VMware Tanzu IDE extension for IntelliJ IDEA. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for IntelliJ currently supports Java applications on macOS and Windows.

Extension features

This extension gives the following features:

- **Deploy applications directly from IntelliJ:**

  Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within IntelliJ.

- **See code updates running on-cluster in seconds:**

  With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

- **See workloads running on the cluster:**

  From the Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Work with microservices in a Java monorepo:**

  Tanzu Developer Tools for IntelliJ v1.3 and later supports working with a monorepo containing multiple modules that represent different microservices. This makes it possible to deploy, debug, and Live Update multiple workloads simultaneously from the same IntelliJ multimodule project. For more information about projects with multiple modules, see the IntelliJ documentation. For more information about a typical monorepo setup, see Working with microservices in a monorepo.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Next steps

Follow the steps to install the extension.

Overview of Tanzu Developer Tools for IntelliJ

Tanzu Developer Tools for IntelliJ is the official VMware Tanzu IDE extension for IntelliJ IDEA. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for IntelliJ currently supports Java applications on macOS and Windows.

Extension features

This extension gives the following features:

- **Deploy applications directly from IntelliJ:**

  Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within IntelliJ.

- **See code updates running on-cluster in seconds:**

  With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

- **See workloads running on the cluster:**

  From the Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Work with microservices in a Java monorepo:**

  Tanzu Developer Tools for IntelliJ v1.3 and later supports working with a monorepo containing multiple modules that represent different microservices. This makes it possible to deploy, debug, and Live Update multiple workloads simultaneously from the same IntelliJ multimodule project. For more information about projects with multiple modules, see the IntelliJ documentation. For more information about a typical monorepo setup, see Working with microservices in a monorepo.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Next steps

Follow the steps to install the extension.

Install Tanzu Developer Tools for IntelliJ

This topic explains how to install the VMware Tanzu Developer Tools for IntelliJ IDE extension. The extension currently only supports Java applications on macOS and Windows.

Prerequisites

Before installing the extension, you must have:

- IntelliJ

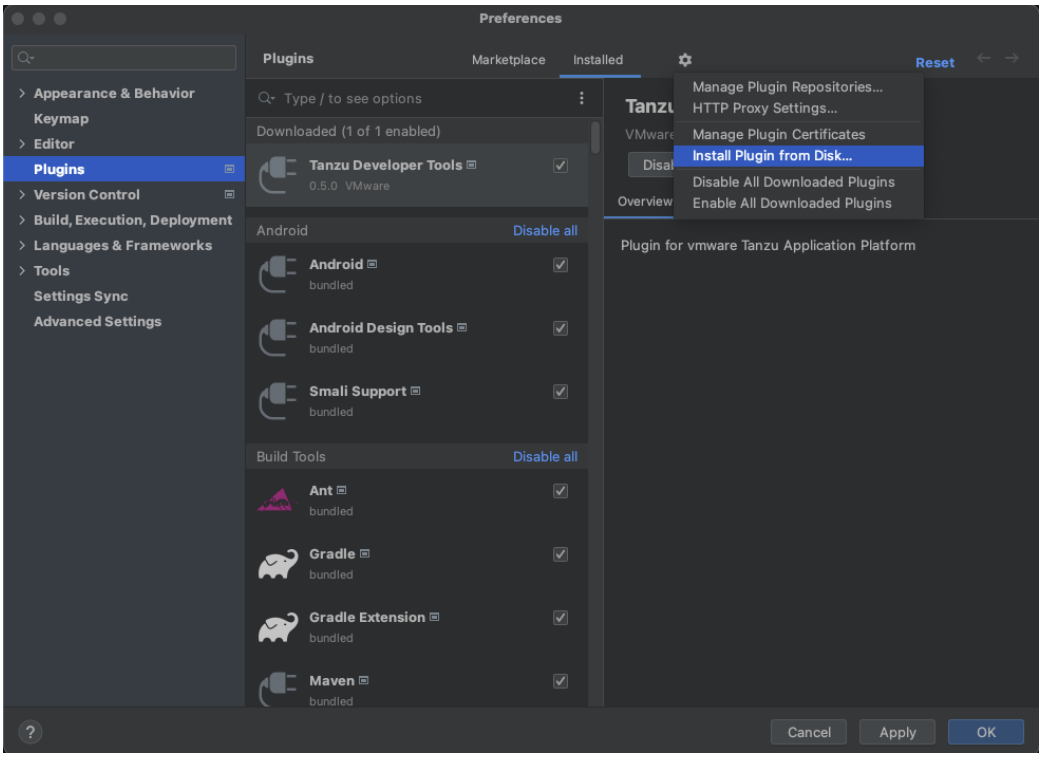
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

> **Note**
>
> If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Install

To install VMware Tanzu Developer Tools for IntelliJ:

1. Download VMware Tanzu Developer Tools for IntelliJ from the VMware Tanzu Network.
2. Open IntelliJ.
3. Open the **Preferences** pane and then go to **Plugins**.
4. Click the gear icon and then click **Install Plugin from disk...**.



5. Use the file picker to select the ZIP file downloaded from the VMware Tanzu Network.

Update

To update to a later version, repeat the steps in the Install section. You do not need to uninstall the current version.

Uninstall

To uninstall the VMware Tanzu Developer Tools for IntelliJ:

1. Open the **Preferences** pane and then go to **Plugins**.
2. Select the extension, click the gear icon, and then click **Uninstall**.
3. Restart IntelliJ.

Next steps

Proceed to Getting started.

Get Started with Tanzu Developer Tools for IntelliJ

This topic guides you through getting started with Tanzu Developer Tools for IntelliJ.

Prerequisite

Install Tanzu Developer Tools for IntelliJ.

Configure source image registry

Before deploying a workload, you must authenticate with an image registry to store your source code. You can use the Docker CLI to authenticate or you can set environment variables that the Tanzu CLI can use to authenticate.

Docker CLI
To authenticate by using the Docker CLI, run:

```
docker login $REGISTRY_HOSTNAME -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
```

Tanzu CLI
To authenticate using the Tanzu CLI, export these environment variables by running:

```
export TANZU_APPS_REGISTRY_CA_CERT=PATH-TO-CA-CERT.nip.io.crt
export TANZU_APPS_REGISTRY_PASSWORD=USERNAME
export TANZU_APPS_REGISTRY_USERNAME=PASSWORD
```

`CA_CERT` is only needed for a custom or private registry.

For more information, see Workload creation fails due to authentication failure in Docker Registry

Run Tanzu Developer Tools for IntelliJ

Run IntelliJ from a CLI, instead of through your operating system GUI, to avoid restricting the set of environment variables the app receives. This is especially relevant for macOS.

Limited environment variables can cause problems with cluster authentication for Tanzu Developer Tools for IntelliJ. For example, a common situation is that a sanitized `PATH` does not provide access to the `gke-cloud-auth-plugin` installed on your system. This makes Tanzu Developer Tools for IntelliJ unable to authenticate and access your GKE cluster.

This situation is complex and different things can go wrong depending on:

- Precisely how you installed various cloud-related CLI tools
- How you set environment variables
- Your OS version

- Which cloud provider and authentication method you are using

All of these problems are most easily avoided by running IntelliJ from a CLI. Run IntelliJ from a CLI in macOS by running:

```
open /Applications/IntelliJ\ IDEA.app
```

Set up Tanzu Developer Tools

The extension makes use of the following files within your project:

- `workload.yaml`
- `catalog-info.yaml`
- `Tiltfile`
- `.tanzuignore`

You can create these files by using the instructions in this topic, or use the files in the View an example project section.

There are two ways to create these files:

- Using the code snippets that Tanzu Developer Tools provide, which create templates in empty files that you then fill in with the required information.
- Writing the files manually.

Create the `workload.yaml` file

You must include a file named `workload.yaml` in your project. For example, `my-project/config/workload.yaml`.

`workload.yaml` provides instructions to Supply Chain Choreographer about how to build and manage a workload. For more information, see Supply Chain Choreographer for Tanzu.

The Tanzu Developer Tools for IntelliJ extension requires only one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multi-document YAML file.

To create a `workload.yaml` file by using code snippets:

1. Right-click on the IntelliJ project explorer and then click **New**.
2. Select the Tanzu workload.
3. Add the filename `workload`.
4. Fill in the template.

See the following `workload.yaml` example:

```
apiVersion: carto.run/v1alpa1
kind: Workload
metadata:
 name: APP-NAME
 labels:
   apps.tanzu.vmware.com/workload-type: WORKLOAD-TYPE
   app.kubernetes.io/part-of: APP-NAME
spec:
 source:
   git:
     url: GIT-SOURCE-URL
     ref:
       branch: GIT-BRANCH-NAME
```

Where:

- `APP-NAME` is the name of your application. For example, `my app`.
- `WORKLOAD-TYPE` is the type of workload for your app. For example, `web`. For more information, see Workload types.
- `GIT-SOURCE-URL` is the Git source code URL for your app. For example, `github.com/mycompany/myapp`.
- `GIT-BRANCH-NAME` is the branch of the Git source code you want to use. For example, `main`.

Alternatively you can use the Tanzu CLI to create a `workload.yaml` file. For more information about the relevant Tanzu CLI command, see Tanzu apps workload apply.

Create the `catalog-info.yaml` file

You must include a file named `catalog-info.yaml` in your project. For example, `my-project/catalog/catalog-info.yaml`.

`catalog-info.yaml` enables the workloads created with Tanzu Developer Tools for IntelliJ to be visible in Tanzu Application Platform GUI. For more information, see Overview of Tanzu Application Platform GUI.

To create a `catalog-info.yaml` file by using the code snippets:

1. Right-click on the IntelliJ project explorer and then click **New**.
2. Select the Tanzu Catalog.
3. Add the filename `catalog-info`.
4. Fill in the template.

See the following `workload.yaml` example:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
 name: APP-NAME
 description: APP-DESCRIPTION
 tags:
   - tanzu
 annotations:
   'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=APP-NAME'
spec:
 type: service
 lifecycle: experimental
 owner: default-team
```

Where:

- `APP-NAME` is the name of your application.
- `APP-DESCRIPTION` is a description of your application.

Create the Tiltfile file

In your project you must include a file named `Tiltfile` with no extension (no filetype), such as `my-project/Tiltfile`.

The `Tiltfile` provides the configuration for Tilt to enable your project to Live Update on the Tanzu Application Platform-enabled Kubernetes cluster. For more information, see the Tilt documentation.

The Tanzu Developer Tools for IntelliJ extension requires only one Tiltfile per project.

The following is an example `Tiltfile`:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='SOURCE-IMAGE-VALUE')
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')

k8s_custom_deploy(
   'APP-NAME',
   apply_cmd="tanzu apps workload apply -f PATH-TO-WORKLOAD-YAML --live-update" +
       " --local-path " + LOCAL_PATH +
       " --source-image " + SOURCE_IMAGE +
       " --namespace " + NAMESPACE +
       " --yes >/dev/null" +
       " && kubectl get workload APP-NAME --namespace " + NAMESPACE + " -o yaml",
   delete_cmd="tanzu apps workload delete -f PATH-TO-WORKLOAD-YAML --namespace " + NAM
ESPACE + " --yes" ,
   deps=['pom.xml', './target/classes'],
   container_selector='workload',
   live_update=[
       sync('./target/classes', '/workspace/BOOT-INF/classes')
   ]
)

k8s_resource('APP-NAME', port_forwards=["8080:8080"],
   extra_pod_selectors=[{'carto.run/workload-name': 'APP-NAME', 'app.kubernetes.io/com
ponent': 'run'}])
allow_k8s_contexts('CONTEXT-NAME')
```

Where:

- `SOURCE-IMAGE-VALUE` is your source image.
- `APP-NAME` is the name of your application.
- `PATH-TO-WORKLOAD-YAML` is the local file system path to your `workload.yaml` file. For example, `config/workload.yaml`.
- `CONTEXT-NAME` is the name of your current Kubernetes context. If your Tanzu Application Platform-enabled Kubernetes cluster is running on your local machine, you can remove the entire `allow_k8s_contexts` line. For more information about this line, see the Tilt documentation.

If you want to compile the source image from a local directory other than the project directory, change the value of `local path`. For more information, see local path in the glossary.

Create the `.tanzuignore` file

In your project, you can include a file named `.tanzuignore` with no file extension. For example, `my-project/.tanzuignore`.

When working with local source code, `.tanzuignore` excludes files from the source code that are uploaded within the image. It has syntax similar to the `.gitignore` file.

For an example, see the `.tanzuignore` file in GitHub that is used for the sample Tanzu Java web app. You can use the file as it is or edit it for your needs.

View an example project

Before you begin, you need a container image registry to use the sample application. There are two ways to view a sample application that demonstrates the necessary configuration files.

Use Application Accelerator

If your company has configured Application Accelerator, you can obtain the sample application there if it was not removed. To view the example using Application Accelerator:

1. Open Application Accelerator. The Application Accelerator location varies based on where your company placed it. Contact the appropriate team to learn its location.
2. Search for `Tanzu Java Web App` in the Application Accelerator.
3. Add the required configuration information and generate the application.
4. Unzip the application and open the directory in IntelliJ.

Clone from GitHub

To clone the example from GitHub:

1. Use `git clone` to clone the application-accelerator-samples repository from GitHub.
2. Go to the `tanzu-java-web-app` directory.
3. Open the `Tiltfile` and replace `your-registry.io/project` with your registry.

Next steps

Use Tanzu Developer Tools for IntelliJ.

Use Tanzu Developer Tools for IntelliJ

Ensure that the project you want to use the Tanzu Developer Tools for IntelliJ extension with has the required files specified in Getting started.

The extension requires only one Tiltfile and one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multi-document YAML file.

Workload Actions

The extension enables you to apply, debug, and Live Update your application on a Kubernetes cluster that has Tanzu Application Platform. The developer sandbox experience enables developers to Live Update their code and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

Apply a workload

The extension enables you to apply workloads on your Kubernetes cluster that has Tanzu Application Platform.

To apply a workload:

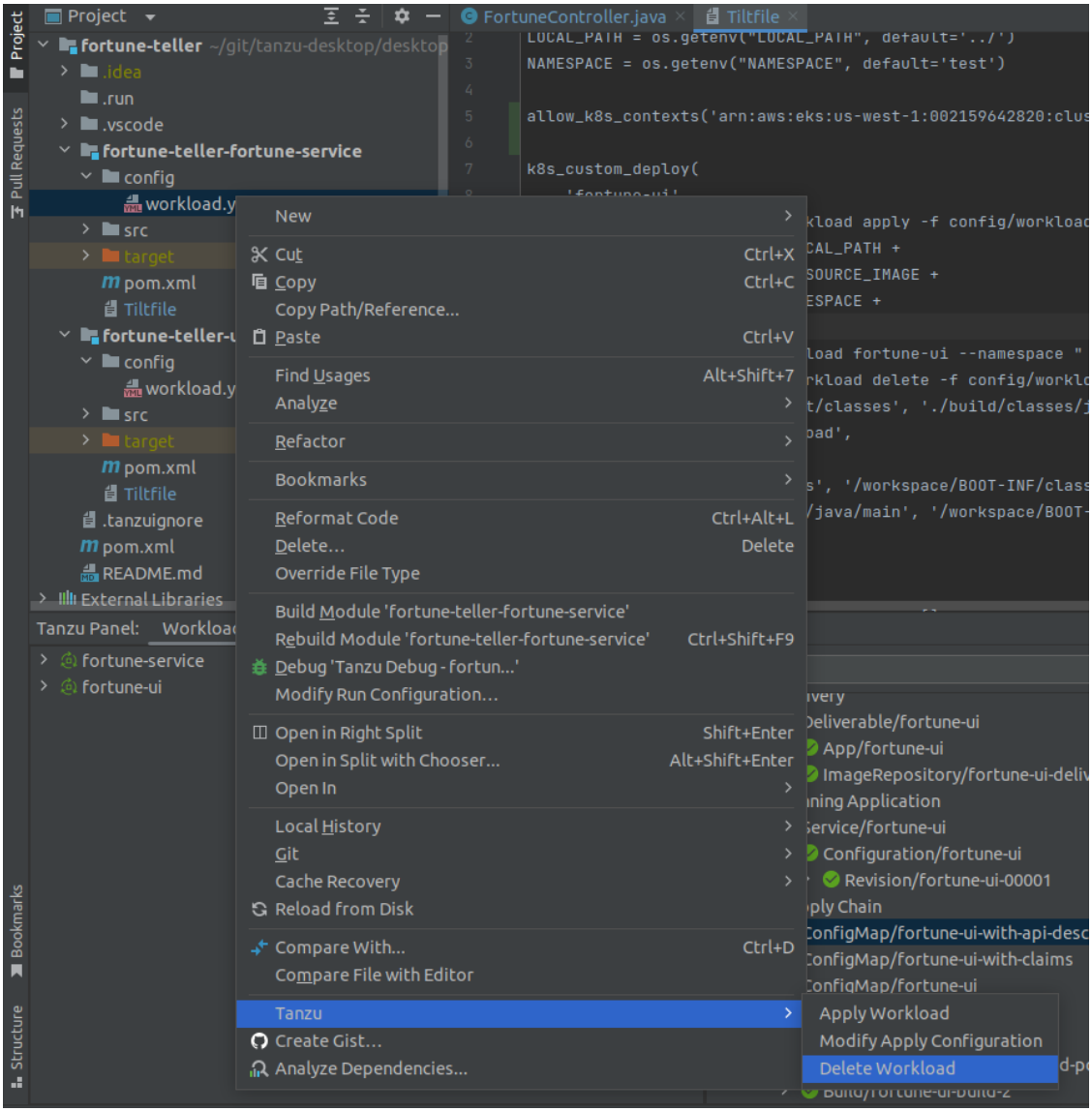
1. Right-click anywhere in the IntelliJ project explorer and click **Tanzu** > **Apply Workload** or right-click on an associated workload in the Workloads panel and click **Apply Workload**.
2. Click **Tanzu > Modify Apply Configuration**.

The `Tanzu workload apply` command is triggered in the terminal and the workload is applied. A new workload appears on the Tanzu panel.

Delete a workload

The extension enables you to delete workloads on your Kubernetes cluster that has Tanzu Application Platform.

To delete a workload right-click anywhere in the IntelliJ project explorer and click **Tanzu** > **Delete Workload** or right-click on an associated workload in the Workloads panel and click **Delete Workload**.



A message appears that prompts you to delete the workload and not warn again, delete the workload, or cancel. A notification appears showing that the workload was deleted.

Debugging on the cluster

The extension enables you to debug your application on a Kubernetes cluster that has Tanzu Application Platform.

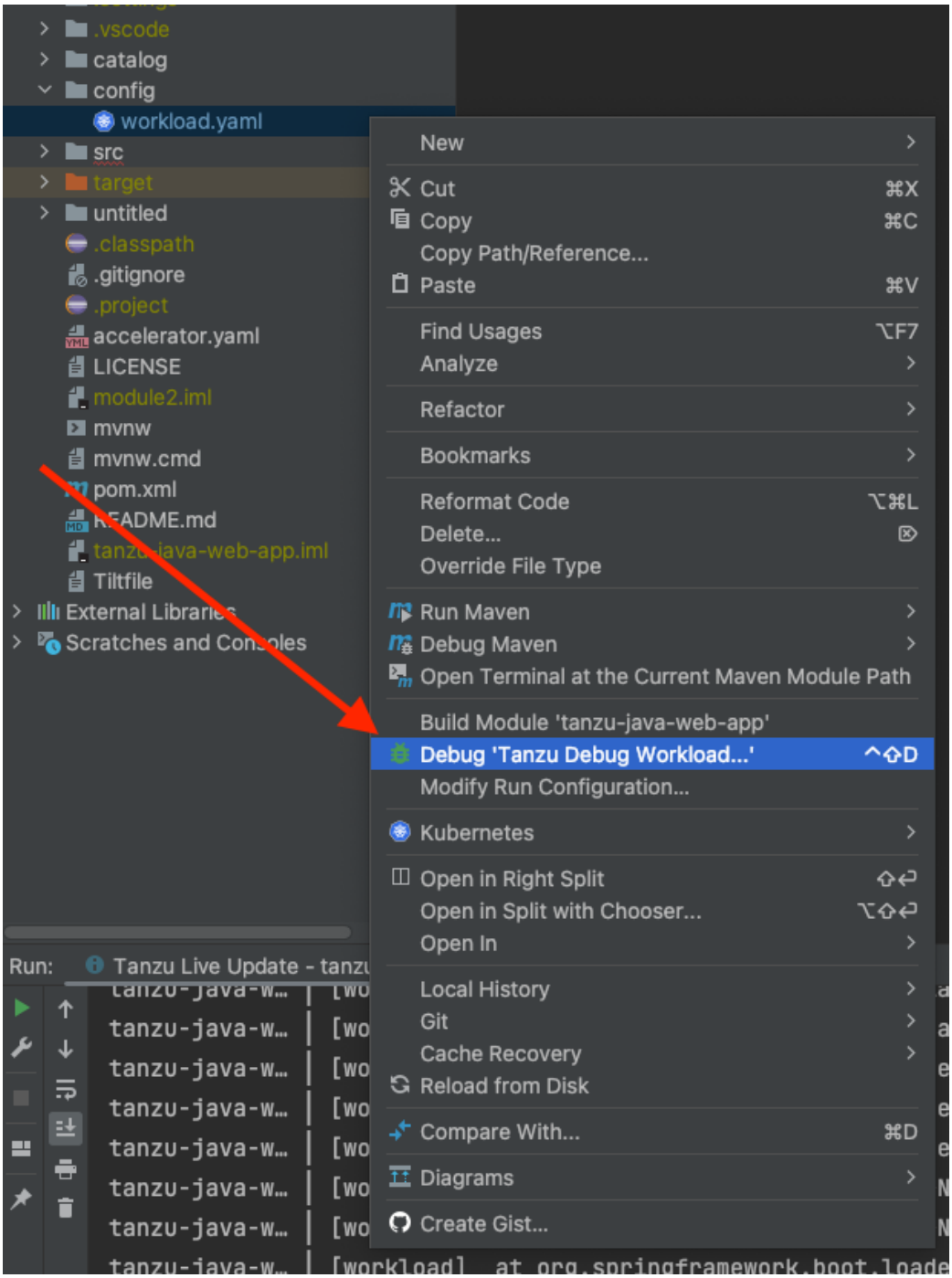
Debugging requires a single-document `workload.yaml` file in your project. For how to create `workload.yaml`, see Set up Tanzu Developer Tools.

The developer sandbox experience enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

Start debugging on the cluster

To start debugging on the cluster:

1. Add a breakpoint in your code.
2. Right-click the `workload.yaml` file in your project and click **Debug 'Tanzu Debug Workload...'** in the pop-up menu or right-click on an associated workload in the Workloads panel and click **Debug Workload**.

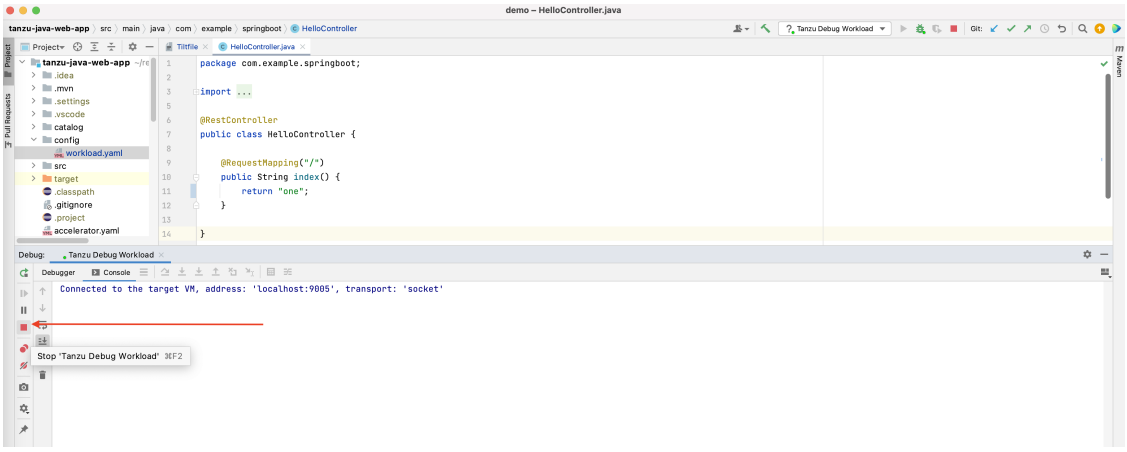


3. Ensure that the configuration parameters are set:
    - **Source Image:** This is the registry location for publishing local source code. For example, `registry.io/yourapp-source`. It must include both a registry and a project name.
    - **Local Path:** This is the path on the local file system to a directory of source code to build.
    - **Namespace:** This is the namespace that workloads are deployed into.

You can also manually create Tanzu Debug configurations by using the **Edit Configurations** IntelliJ UI.

Stop Debugging on the Cluster

Click the stop button in the **Debug** overlay to stop debugging on the cluster.



Live Update

See the following sections for how to use Live Update.

Start Live Update

Before using Live Update, verify that your auto-save setting is either off or on with a delay. The delay must be long enough for the application to restart between auto saves to allow enough time for your app to Live Update when files change. This auto-save setting is in **Preferences** > **Appearance & Behavior** > **System Settings** > **Autosave**.

To start Live Update:

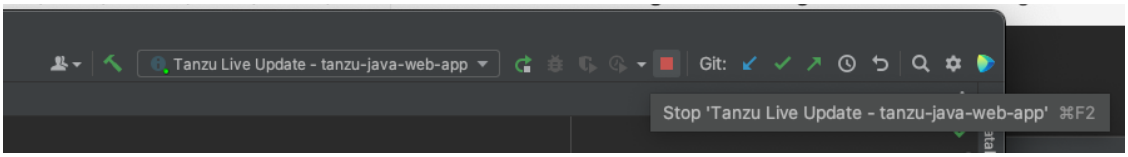
1. Right-click your project's Tiltfile and then click **Run 'Tanzu Live Update - ...'** or right-click on an associated workload in the Workloads panel and then click **Live Update Workload**.
2. Ensure that the configuration parameters are set:
    - **Source Image:** This is the registry location for publishing local source code. For example, `registry.io/yourapp-source`. It must include both a registry and a project name.
    - **Local Path:** This is the path on the local file system to a directory of source code to build.
    - **Namespace:** This is the namespace that workloads are deployed into.

Note

You must compile your code before the changes are synchronized to the container. For example, `Build Project`: `⌘+F9`.

Stop Live Update

To stop Live Update, use the native controls to stop the Tanzu Live Update Run Configuration that is running.



Tanzu Workloads panel

The current state of the workloads is visible in the Tanzu Workloads view. This view is a separate section in the bottom of the Explorer view in the Side Bar. The view shows the current status of each workload, namespace, and cluster. It also shows whether Live Update and Debug is running, stopped, or deactivated.

The Tanzu Activity tab in the Panels view enables developers to visualize the supply chain, delivery, and running application pods. The tab enables a developer to view and describe logs on each resource associated with a workload from within their IDE. The tab displays detailed error messages for each resource in an error state.

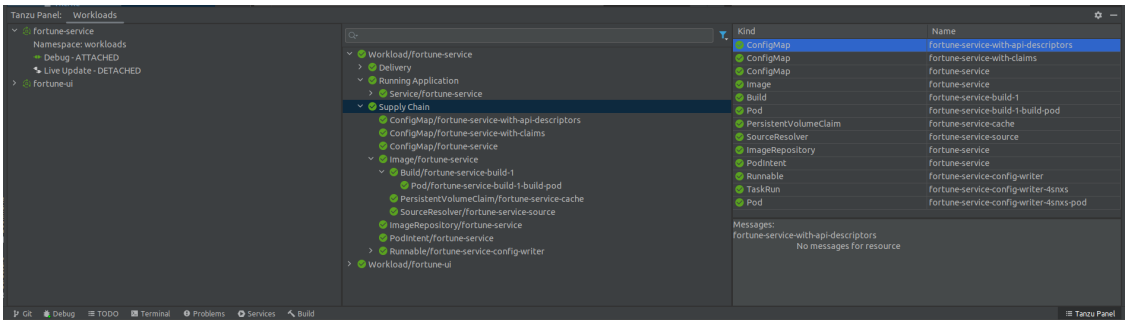
Workload commands are available from the Tanzu Workloads panel on workloads that have an associated module in the current project.

This association is based on a module name and a workload name matching. For example, a project with a module named `my-app` is associated with a deployed workload named `my-app`.

When taking an action from the Tanzu Workloads panel, the action uses the namespace of the deployed workload regardless of the configuration in the module.

For example, you might have a Live Update configuration with a namespace argument of `my-apps-1`, but running the action from a deployed workload in namespace `my-apps-2` starts a Live Update session with a namespace argument of `my-apps-2`.

The Tanzu Workloads panel uses the cluster and defaults to the namespace specified in the current kubectl context.



To add a namespace:

1. View the current context and namespace by running:

   ```
   kubectl config get-contexts
   ```

2. Set a namespace for the current context by running:

   ```
   kubectl config set-context --current --namespace=YOUR-NAMESPACE
   ```

To add additional namespaces to your Workloads panel:

1. Click on the gear icon in the upper right corner of the Workloads panel.
2. Click on **Select Namespaces...**
3. Select the checkboxes of the namespaces that you want to add to your panel.

Working with microservices in a monorepo

A monorepo is single Git repository that contains multiple workloads. Each individual workload is placed in a subfolder of the main repository.

You can find an example of this in Application Accelerator.

The relevant accelerator is called Spring SMTP Gateway, and you can obtain its source code as an accelerator or directly from the application-accelerator-samples GitHub repository.

This project is an example of a typical layout:

- `MONO-REPO-ROOT/`
  - `pom.xml` (parent pom)
  - `microservice-app-1/`
  - `pom.xml`
  - `mvnw` (and other mvn-related files for building the workload)
  - `Tiltfile` (supports Live Update)
  - `config`
    - `workload.yaml` (supports deploying and debugging from IntelliJ)
  - `src/` (contains source code for this microservice)
  - `microservice-app-2/`
  - …similar layout

Recommended structure: Microservices that can be built independently

In this example, each of the microservices can be built independently of one another. Each subfolder contains everything needed to build that workload.

This is reflected in the `source` section of `workload.yaml` by using the `subPath` attribute:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: microservice-app-1
  ...
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/sample-mono-repo.git
    subPath: microservice-app-1 # build only this
  ...
```

For setting up your own repositories, it's best practice to set up a monorepo so that each microservice can be built completely independently.

To work with these monorepos:

1. Import the monorepo as a project into IntelliJ.
2. Interact with each of the subfolders as you would interact with a project containing a single workload.

Alternative structure: Services with build-time interdependencies

Some monorepos do not have submodules that can be independently built. Instead the `pom.xml` files of the submodules are set up to have some build-time interdependencies. For example:

- A submodule `pom.xml` can reference the parent `pom.xml` as a common place for centralized dependency management.
- A microservice submodule can reference another, as a maven dependency.
- Several microservice submodules can reference one or more shared library modules.

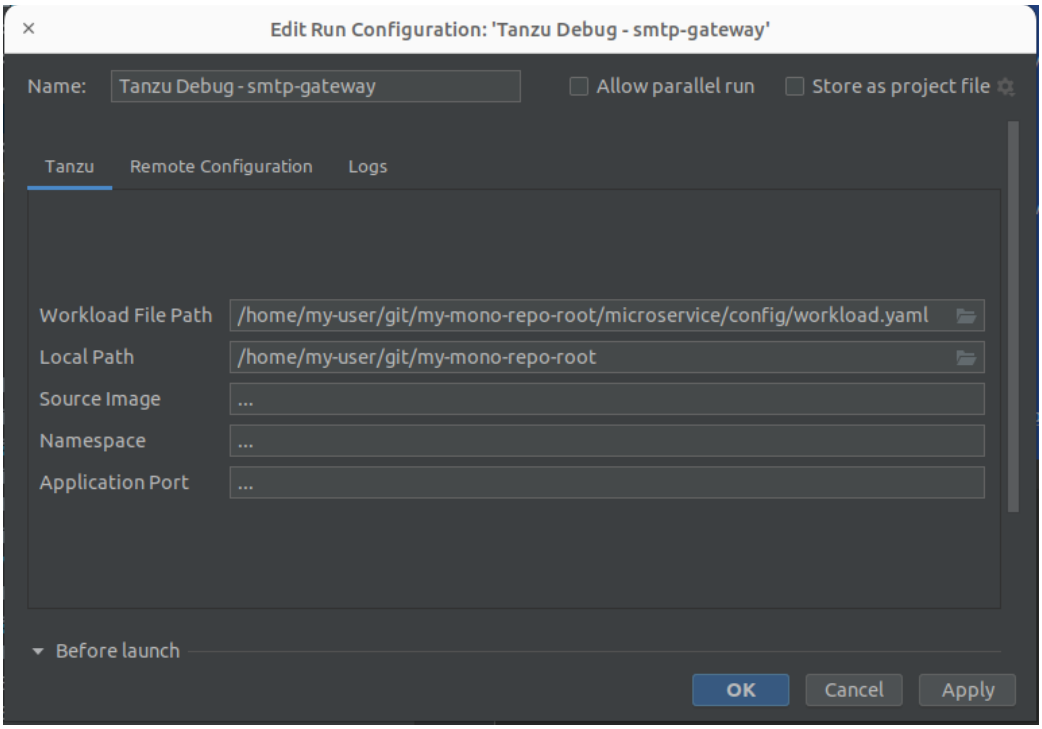
For these projects, make these adjustments:

1. Make `workload.yaml` point to the repository root, not a subfolder. Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.
2. Make `workload.yaml` specify additional buildpack arguments through environment variables. They differentiate the submodule that the build is targeting.

   Both of these `workload.yaml` changes are in the following example:

   ```
   apiVersion: carto.run/v1alpha1
   kind: Workload
   metadata:
   name: fortune-ui
   labels:
     apps.tanzu.vmware.com/workload-type: web
     app.kubernetes.io/part-of: fortune-ui
   spec:
   build:
     env:
        - name: BP_MAVEN_BUILD_ARGUMENTS
        value: package -pl fortune-teller-ui -am # indicate which module to build.
        - name: BP_MAVEN_BUILT_MODULE
        value: fortune-teller-ui # indicate where to find the built artefact to de
   ploy.
   source:
     git:
        url: https://github.com/my-user/fortune-teller # repository root
        ref:
        branch: main
   ```

   For more information about these and other `BP_xxx` buildpack parameters, see the Buildpack documentation.
3. Make the local path attribute in the launch configuration for each workload point to the path of the repository root. Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.



Changing logging verbosity

The Tanzu Language Server saves logs to `~/tanzu-langserver.log`. You can change the log verbosity in **Preferences** > **Tools** > **Tanzu Developer Tools**.

Glossary of terms

This topic gives you explanations of common terms used throughout the Tanzu Developer Tools for IntelliJ documentation, and within the extension itself. Some of these terms are unique to Tanzu Application Platform, while others might have a different meaning outside of Tanzu Application Platform and are included here for clarification.

Live Update

Live Update, facilitated by Tilt, enables you to deploy your workload once, save changes to the code, and see those changes reflected in the workload running on the cluster within seconds.

Tiltfile

The Tiltfile is a file with no extension that is required for Tilt to enable the Live Update feature. For more information about the Tiltfile, see the Tilt documentation.

Debugging on the cluster

The Tanzu Developer Tools on IntelliJ extension enables you to debug your application in an environment similar to production by debugging on your Tanzu Application Platform enabled Kubernetes cluster.

Note

An environment's similarity to production relies on keeping dependencies updated, among other variables.

YAML file format

YAML is a human-readable data-serialization language. It is commonly used for configuration files. For more information, see the YAML Wikipedia entry.

workload.yaml file

The workload YAML file is a required configuration file used by the Tanzu Application Platform to specify the details of an application including its name, type, and source code URL.

catalog-info.yaml file

The catalog-info YAML file enables the workloads created with the Tanzu Developer Tools for IntelliJ extension to be visible in the Tanzu Application Platform GUI.

Code snippet

Code snippets enable you to quickly add project files that are necessary to develop using Tanzu Application Platform by creating a template in an empty file that you fill out with the required information.

Source image

The source image is the registry location to publish local source code, for example, `registry.io/yourapp-source`. This must include both a registry and a project name.

Local path

The local path value tells the Tanzu Developer Tools for IntelliJ extension which directory on your local file system to bring into the source image container image. The default local path value is the current directory where you saved the files for your open IntelliJ project.

Kubernetes context

A Kubernetes context is a set of access parameters that contains a Kubernetes cluster, a user, and a namespace. A Kubernetes context acts like a set of coordinates that describe the target of the Kubernetes commands that you run. For more information, see the Kubernetes documentation.

Kubernetes namespace

As defined by the Kubernetes documentation, in Kubernetes, namespaces provide a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace, but not across namespaces.

Troubleshoot Tanzu Developer Tools for IntelliJ

This topic helps you troubleshoot issues with Tanzu Developer Tools for IntelliJ.

Tanzu Debug re-applies the workload when namespace field is empty

Symptoms

If the `namespace` field of the debug launch configuration is empty, the workload is re-applied even if it exists on the cluster.

Cause

Internally, workloads are gathered in the cluster in the current namespace and compared with the information that you specify. If the `namespace` field is empty, it is considered `null` and the internal checks fail.

Solution

Do not leave the `namespace` field blank.

Workload is wrongly re-applied because of debug configuration selected from the launch configuration drop-down menu

Symptoms

If your debug configuration is created from the launch configuration drop-down menu, it re-applies the workload even if the workload already exists on the cluster.

Cause

There is internal logic that is not run when debug configuration is created from the drop-down menu. However, the logic is run when debug configuration is selected from the right-click pop-up menu.

Solution

Select debug configuration from the right-click pop-up menu.

Unable to view workloads on the panel when connected to GKE cluster

Symptom

When connecting to Google's GKE clusters, an error appears with the text `WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.25+; use gcloud instead.`

Cause

GKE authentication was extracted into a separate plug-in and is no longer inside the Kubernetes client or libraries.

Solution

Download and configure the GKE authentication plug-in. For instructions, see the Google documentation.

Deactivated launch controls after running a launch configuration

Symptom

When you run or debug a launch configuration, IntelliJ deactivates the launch controls.

Cause

IntelliJ deactivates the launch controls to prevent other launch configurations from being launched at the same time. These controls are reactivated when the launch configuration is started. As such, starting multiple Tanzu debug and live update sessions is a synchronous activity.

Starting a Tanzu Debug session fails with `Unable to open debugger port`

Symptom

You try to start a Tanzu Debug session and it immediately fails with an error message similar to:

```
Error running 'Tanzu Debug - fortune-teller-fortune-service': Unable to open debugger
port (localhost:5005): java.net.ConnectException "Connection refused"
```

Cause

Old Tanzu Debug launch configurations sometimes appear to be corrupted after installing a later version of the plug-in. You can see whether this is the problem you are experiencing by opening the launch configuration:

1. Right-click `workload.yaml`.
2. Click **Modify Run Configuration...** in the menu.
3. Scroll down and expand the **Before Launch** section of the dialog.
4. Verify that it contains the two Unknown Task entries `com.vmware.tanzu.tanzuBeforeRunPortForward` and `com.vmware.tanzu.tanzuBeforeRunWorkloadApply`.

Because these two tasks are unknown causes, these steps of the debug launch are not run. This in turn means that the target application is not deployed and accessible on the expected port, which causes an error when the debugger tries to connect to it.

It might be that although the launch configuration appears corrupt when seen in the launch config editor, in fact there is no corruption. It's suspected that this problem only occurs when you install a new version of the plug-in and start using it before first restarting IntelliJ.

There is possibly an issue in the IntelliJ platform that prevents completely or correctly initializing the plug-in when the plug-in is hot-swapped into an active session instead of loaded on startup.

Solution

Closing and restarting IntelliJ typically fixes this problem. If that doesn't work for you, delete the old corrupted launch configuration and recreate it.

Timeout error when Live Updating

Symptom

When you attempt to Live Update your workload, the following error message appears in the log:

```
ERROR: Build Failed: apply command timed out after 30s - see }}{{https://docs.tilt.dev/api.html#api.update_settings{{ for how to increase}}
```

Cause

Kubernetes times out on upserts over 30 seconds.

Solution

Add `update_settings (k8s_upsert_timeout_secs = 300)` to the Tiltfile. For more information, see the Tiltfile documentation.

Tanzu Panel empty when using a GKE cluster on macOS

Symptom

On macOS, the Tanzu Panel doesn't display workloads or any other resources when using a GKE cluster. Other tools, such as the Tanzu CLI Apps plug-in, display resources correctly.

Cause

`gke-cloud-auth-plugin` is required to properly authenticate to a GKE cluster. However, when starting IntelliJ from Dock or Spotlight, environment variables set by using `.profile`, `.bash_profile`, or `.zshrc` are not available. For more information, see this YouTrack issue.

This might cause `gke-cloud-auth-plugin` to be missing from `PATH` when launching IntelliJ and prevent the Tanzu Panel from reaching the cluster.

Solution

Open IntelliJ from the CLI. Example command:

```
open /Applications/IntelliJ\ IDEA.app
```

Tanzu panel shows workloads but doesn't show Kubernetes resources

Symptom

The Tanzu panel shows workloads but doesn't show Kubernetes resources in the center panel of the activity pane.

Cause

When switching the Kubernetes context, the activity pane doesn't automatically update the namespace, but the workload pane detects the new namespace. Therefore, the Tanzu panel shows workloads but doesn't show Kubernetes resources in the center panel of the activity pane.

Solution

Restart IntelliJ to properly detect the context change.

Tanzu Workloads panel workloads only have describe and delete action

Symptom

Some or all workloads in the Tanzu Workloads panel only have describe and delete actions.

Cause

By design, only associated workloads have apply, debug, and Live Update workload actions available.

Solution

Open a project that contains a module that can be associated with your deployed workloads.

Workload actions do not work when in a project with spaces in the name

Symptom

Workload actions do not work. The console displays an error message similar to the following:

```
Error: unknown command "projects/my-app" for "apps workload apply"Process finished wit
h exit code 1
```

Cause

On Windows, workload actions do not work when in a project with spaces in the name, such as `my-app project`.

Solution

1. Close the code editor.
2. Move or rename your project folder on the disk, ensuring that no part of its path contains any spaces.
3. Delete the project settings folder from the project to start with a clean slate. The folder is `.idea` if using IntelliJ and `.vscode` if using VS Code.
4. Open the code editor and then open the project in its new location.

`config-writer-pull-requester` is categorized as Unknown

Symptom

In the Tanzu Activity Panel, the `config-writer-pull-requester` of type `Runnable` is incorrectly categorized as **Unknown**. It should be in the **Supply Chain** category.

Solution

A fix is planned for a future release.

Frequent application restarts

Symptom

When an application is applied from IntelliJ it restarts frequently.

Cause

An application or environment behavior is triggering the application to restart.

Observed trigger behaviors include:

- The application itself writing logs to the file system in the application directory that Live Update is watching
- Autosave being set to a very high frequency in the IDE configuration

Solution

Prevent the trigger behavior. Example solutions include:

- Prevent 12-factor applications from writing to the file system.
- Reduce the autosave frequency to once every few minutes.

Overview of Tanzu Developer Tools for Visual Studio

VMware Tanzu Developer Tools for Visual Studio is the official VMware Tanzu IDE extension for Visual Studio 2022. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for Visual Studio currently supports .NET C# applications.

This extension is for Microsoft Visual Studio 2022 only. It is incompatible with Visual Studio Code and Visual Studio for Mac.

Note

This extension is in the beta stage of development.

Extension features

The extension has the following features:

- **Deploy applications directly from Visual Studio:**

  Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within Visual Studio.

- **See code updates running on-cluster in seconds:**

  With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

- **See workloads running on the cluster:**

From the Tanzu Panel, you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Work with microservices in a Visual Studio solution:**

  Work with multiple solution projects that represent discrete microservices. This makes it possible to deploy, debug, and Live Update multiple workloads simultaneously from the same solution.

> **Note**
>
> The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Next steps

Install Tanzu Developer Tools for Visual Studio.

Overview of Tanzu Developer Tools for Visual Studio

VMware Tanzu Developer Tools for Visual Studio is the official VMware Tanzu IDE extension for Visual Studio 2022. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for Visual Studio currently supports .NET C# applications.

This extension is for Microsoft Visual Studio 2022 only. It is incompatible with Visual Studio Code and Visual Studio for Mac.

> **Note**
>
> This extension is in the beta stage of development.

Extension features

The extension has the following features:

- **Deploy applications directly from Visual Studio:**

  Rapidly iterate on your applications on Tanzu Application Platform and deploy them as workloads directly from within Visual Studio.

- **See code updates running on-cluster in seconds:**

  With the use of Live Update facilitated by Tilt, deploy your workload once, save changes to the code and then, seconds later, see those changes reflected in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies updated, among other variables.

- **See workloads running on the cluster:**

From the Tanzu Panel, you can see any workload found within the cluster and namespace specified in the current kubectl context.

- **Work with microservices in a Visual Studio solution:**

  Work with multiple solution projects that represent discrete microservices. This makes it possible to deploy, debug, and Live Update multiple workloads simultaneously from the same solution.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Next steps

Install Tanzu Developer Tools for Visual Studio.

Install Tanzu Developer Tools for Visual Studio

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio.

Prerequisites

Before installing the extension, you must have:

- Visual Studio 2022 v17.7 or later
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

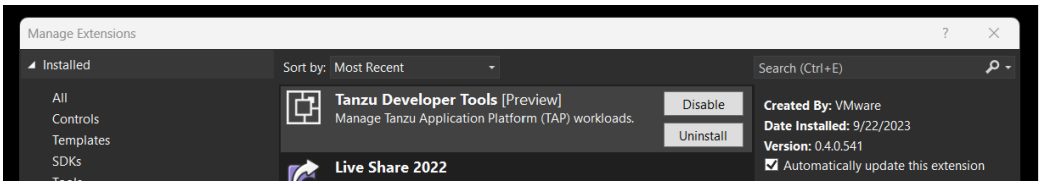
Note

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

Install

To install Tanzu Developer Tools for Visual Studio:

1. Download VMware Tanzu Developer Tools for Visual Studio from VMware Tanzu Network.
2. Double-click the `.vsix` install file and click through the prompts.
3. Open Visual Studio and, from top menu, click **Extensions** > **Manage Extensions**.
4. Verify that the extension is installed and that it is the version you want.



Update

To update to a later version, repeat the steps in the Install section. You do not need to uninstall the current version.

Uninstall

To uninstall:

1. From the top menu, click the **Extensions** tab and then click **Manage Extensions**.
2. Select the **Installed** section and then click the **Uninstall** button for this extension.

Next steps

Getting Started with Tanzu Developer Tools for Visual Studio.

Get Started with Tanzu Developer Tools for Visual Studio

This topic guides you through getting started with VMware Tanzu Developer Tools for Visual Studio.

Prerequisite

Install Tanzu Developer Tools for Visual Studio.

Configure source image registry

Before deploying a workload, you must authenticate with an image registry to store your source code. You can use the Docker CLI to authenticate or you can set environment variables that the Tanzu CLI can use to authenticate.

Docker CLI
To authenticate by using the Docker CLI, run:

```
docker login $REGISTRY_HOSTNAME -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
```

Tanzu CLI
To authenticate using the Tanzu CLI, export these environment variables by running:

```
export TANZU_APPS_REGISTRY_CA_CERT=PATH-TO-CA-CERT.nip.io.crt
export TANZU_APPS_REGISTRY_PASSWORD=USERNAME
export TANZU_APPS_REGISTRY_USERNAME=PASSWORD
```

`CA_CERT` is only needed for a custom or private registry.

For more information, see Workload creation fails due to authentication failure in Docker Registry

Set up Tanzu Developer Tools

The extension makes use of the following files within your project:

- `workload.yaml`
- `catalog-info.yaml`
- `Tiltfile`
- `.tanzuignore`

You can create these files by using the instructions in this topic, or use the files in the View an example project section.

There are two ways to create these files:

- Using the code snippets that Tanzu Developer Tools provide, which create templates in empty files that you then fill in with the required information.
- Writing the files manually.

Create the `workload.yaml` file

Your project must contain a file named `workload.yaml`. For example, `MyApp\Config\workload.yaml`.

`workload.yaml` provides instructions to Supply Chain Choreographer for how to build and manage a workload. For more information, see Supply Chain Choreographer for Tanzu.

The Tanzu Developer Tools for Visual Studio extension requires at least one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multi-document YAML file.

To create a `workload.yaml` file by using Visual Studio:

1. Right-click the Solution Explorer project.
2. Click **Add** > **New Folder**.
3. Name the folder `Config`.
4. Right-click the new `Config` folder and then click **Add** > **New Item...**.
5. From the available list of items, click **Tanzu Workload** > **Add**.
6. Follow the instructions at the top of the created file.

See the following `workload.yaml` example:

```
apiVersion: carto.run/v1alpa1
kind: Workload
metadata:
 name: APP-NAME
 labels:
   apps.tanzu.vmware.com/workload-type: WORKLOAD-TYPE
   app.kubernetes.io/part-of: APP-NAME
spec:
 source:
   git:
     url: GIT-SOURCE-URL
     ref:
       branch: GIT-BRANCH-NAME
```

Where:

- `APP-NAME` is the name of your application. For example, `my-app`.
- `WORKLOAD-TYPE` is the type of workload for your app. For example, `web`. For more information, see Workload types.

- `GIT-SOURCE-URL` is the Git source code URL for your app. For example, `github.com/mycompany/myapp`.
- `GIT-BRANCH-NAME` is the branch of the Git source code you want to use. For example, `main`.

Alternatively, you can use the Tanzu CLI to create a `workload.yaml` file. For more information about the relevant Tanzu CLI command, see Tanzu apps workload apply.

Create the `catalog-info.yaml` file

Your project must contain a file named `catalog-info.yaml`. For example, `MyApp\Catalog\catalog-info.yaml`.

`catalog-info.yaml` enables the workloads created with Tanzu Developer Tools for Visual Studio to appear in Tanzu Application Platform GUI. For more information, see Overview of Tanzu Application Platform GUI.

To create a `catalog-info.yaml` file by using Visual Studio:

1. Right-click the Solution Explorer project.
2. Click **Add** > **New Folder**.
3. Name the folder `Catalog`.
4. Right-click the new `Catalog` folder and then click **Add** > **New Item...**.
5. From the available list of items, click **Tanzu Catalog Info** > **Add**.
6. Follow the instructions at the top of the created file.

See the following `catalog-info.yaml` example:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
 name: APP-NAME
 description: APP-DESCRIPTION
 tags:
   - tanzu
 annotations:
   'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=APP-NAME'
spec:
 type: service
 lifecycle: experimental
 owner: default-team
```

Where:

- `APP-NAME` is the name of your application.
- `APP-DESCRIPTION` is a description of your application.

Create the Tiltfile file

Your project must contain a file named `Tiltfile`. For example, `MyApp\Tiltfile`.

The `Tiltfile` provides the configuration for Tilt to enable your project to Live Update on the Tanzu Application Platform-enabled Kubernetes cluster. For more information, see the Tilt documentation.

To create a `Tiltfile` file by using Visual Studio:

1. Right-click the Solution Explorer project.
2. Click **Add** > **New Item...**

3. From the available list of items, click **Tanzu Tiltfile** > **Add**.
4. Follow the instructions at the top of the created file.

See the following `Tiltfile` example:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='SOURCE-IMAGE-VALUE')
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
NAMESPACE = os.getenv("NAMESPACE", default='default')
LIVE_UPDATE_PATH = os.getenv("LIVE_UPDATE_PATH", default='bin/Debug/net6.0')

k8s_custom_deploy(
   'APP-NAME',
   apply_cmd="tanzu apps workload apply -f Config/workload.yaml --live-update" +
       " --local-path " + LOCAL_PATH +
       " --source-image " + SOURCE_IMAGE +
       " --build-env BP_DEBUG_ENABLED=true" +
       " --namespace " + NAMESPACE +
       " --output yaml" +
       " --yes",
   delete_cmd="tanzu apps workload delete " +  APP-NAME + " --namespace " + NAMESPACE
+ " --yes" ,
   deps=['bin'],
   container_selector='workload',
   live_update=[
       sync(LIVE_UPDATE_PATH, '/workspace')
   ]
)

k8s_resource('APP-NAME', port_forwards=["8080:8080"],
   extra_pod_selectors=[{'carto.run/workload-name': 'APP-NAME', 'app.kubernetes.io/com
ponent': 'run'}])
allow_k8s_contexts('CONTEXT-NAME')
```

Where:

- `SOURCE-IMAGE-VALUE` is your source image
- `APP-NAME` is the name of your application

If your Tanzu Application Platform-enabled Kubernetes cluster is running on your local machine, you can remove the entire `allow_k8s_contexts` line. For more information about this line, see the Tilt documentation.

Create the `.tanzuignore` file

Your project can contain a file named `.tanzuignore`. When working with local source code, `.tanzuignore` excludes files from the source code that is uploaded within the image. It has syntax similar to the `.gitignore` file.

This file must be placed in the project root to work. For example, `MyApp\.tanzuignore`.

To create a `Tiltfile` file by using Visual Studio:

1. Right-click the Solution Explorer project.
2. Click **Add** > **New Item...**.
3. From the available list of items, click **Tanzu Ignore file** > **Add**.

For an example, see the `.tanzuignore` file in GitHub that is used for the sample Tanzu Java web app. You can use the file as it is or edit it for your needs.

View an example project

Before you begin, you need a container image registry to use the sample application. There are two ways to view a sample application that demonstrates the necessary configuration files:

Use Application Accelerator
If your company has configured Application Accelerator, you can obtain the sample application there if it was not removed.

To view the example by using Application Accelerator:

1. Open Application Accelerator. You might need to contact a separate team in your organization to learn they placed it.
2. Search for `Steeltoe Weather Forecast` in Application Accelerator.
3. Add the required configuration information and generate the application.
4. Unzip the application and open the directory in Visual Studio.

Clone from GitHub
To clone the example from GitHub:

1. Use `git clone` to clone the application-accelerator-samples repository from GitHub.
2. Go to the `weatherforecast-steeltoe` directory.
3. Open the `Tiltfile` and replace `your-registry.io/project` with your registry.

Next steps

Use Tanzu Developer Tools for Visual Studio.

Use Tanzu Developer Tools for Visual Studio

This topic tells you how to use VMware Tanzu Developer Tools for Visual Studio.

Note

This extension is in the beta stage of development.

Configure settings

To configure settings, right-click anywhere in the Solution Explorer and click **Tanzu** > **Settings...**.

- Tanzu CLI is installed in a location in your `PATH` environment variable.
- A valid `workload.yaml` file is in the project. For more information, see the specification for Tanzu apps workload apply.
- You have a functional Tanzu Application Platform environment.
- Your kubeconfig file is modified for Tanzu Application Platform workload deployments.
- You have an image repository to which source code in the local file system can be uploaded before Build Service builds it.

Workload Actions

The extension enables you to apply, debug, and Live Update your application on a Kubernetes cluster that has Tanzu Application Platform. The developer sandbox experience enables you to Live Update your code, and simultaneously debug the updated code, without deactivating Live Update.

Apply a workload

To apply a workload, right-click anywhere in the Solution Explorer and click **Tanzu** > **Apply Workload**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Apply Workload**.

Delete a workload

To delete a workload, right-click anywhere in the Solution Explorer and click **Tanzu** > **Delete Workload**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Delete Workload**.

Start debugging on the cluster

To remote debug a workload, right-click anywhere in the Solution Explorer and click **Tanzu** > **Debug Workload**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Debug Workload**.

Caution

Do not use the red square Stop button to end your debugging session. Using the red square Stop button might cause the Tanzu Application Platform workload to fail. Instead, in the top menu click **Debug** > **Detach All**.

If the name of your running app process (the app DLL process), does not match the name of your .NET project as shown in the Visual Studio Solution Explorer, the remote debugging agent might fail to attach.

Live Update

See the following sections for how to use Live Update.

Start Live Update

Ensure that the following Tanzu Settings parameters are set:

- **Local Path**, which is the path on the local file system to a directory of source code to build.
- **Namespace**, which is the namespace that workloads are deployed into. Optional.
- **Source Image**, which is the registry location for publishing local source code. For example, `registry.io/yourapp-source`. It must include both a registry and a project name. The source image parameter is not needed if you configured Local Source Proxy.

To start Live Update, right-click anywhere in the Solution Explorer and click **Tanzu** > **Start Live Update**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Start Live Update**.

After starting Live Update, local builds changes are synchronized with the container.

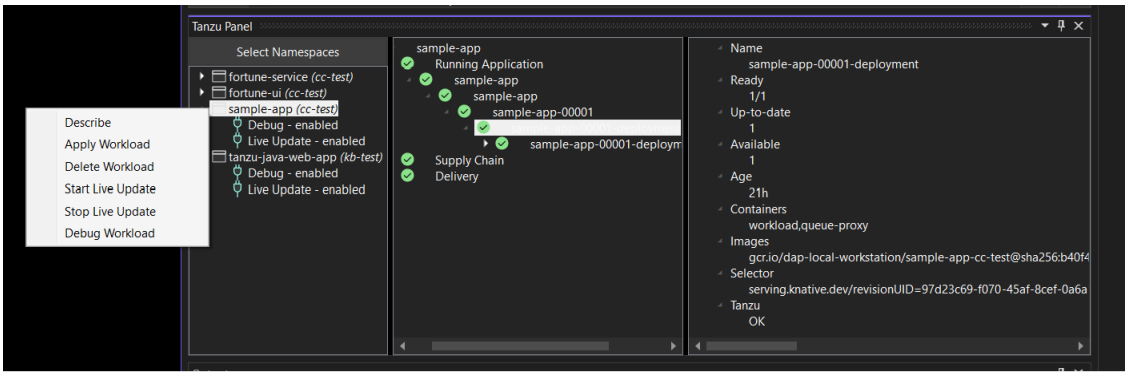
Stop Live Update

To stop Live Update, right-click anywhere in the Solution Explorer and click **Tanzu** > **Stop Live Update**. Alternatively, right-click an associated workload in the Tanzu Panel and click **Stop Live Update**.

Tanzu Workloads panel

The extension creates log entries in a file named `tanzu-dev-tools.log`. This file is in the directory where Visual Studio Installer installed the extension.

To view the Tanzu Workloads panel, right-click anywhere in the Solution Explorer and click **Tanzu** > **View Workloads**.



Extension logs

The extension creates log entries in two files named `tanzu-dev-tools-{GUID}.log` and `tanzu-language-server-{GUID}.log`. These files are in the directory where Visual Studio Installer installed the extension.

To find the log files from PowerShell, run:

```
dir $Env:LOCALAPPDATA\Microsoft\VisualStudio\*\Extensions\*\Logs\tanzu-*.log
```

To find the log files from CMD, run:

```
dir %LOCALAPPDATA%\Microsoft\VisualStudio\*\Extensions\*\Logs\tanzu-*.log
```

Troubleshoot Tanzu Developer Tools for Visual Studio

This topic tells you how to troubleshoot issues you encounter with VMware Tanzu Developer Tools for Visual Studio.

Stop button causes workload to fail

Symptom

Clicking the red square Stop button in the Visual Studio top toolbar causes the Tanzu Application Platform workload to fail or become unresponsive indefinitely.

Solution

To end a debugging session, in the top menu click **Debug** > **Detach All**.

Frequent application restarts

Symptom

When an application is applied from Visual Studio it restarts frequently.

Cause

An application or environment behavior is triggering the application to restart.

Observed trigger behaviors include:

- The application itself writing logs to the file system in the application directory that Live Update is watching
- Autosave being set to a very high frequency in the IDE configuration

Solution

Prevent the trigger behavior. Example solutions include:

- Prevent 12-factor applications from writing to the file system.
- Reduce the autosave frequency to once every few minutes.

Overview of Tanzu Developer Tools for VS Code

VMware Tanzu Developer Tools for Visual Studio Code (VS Code) is the official VMware Tanzu IDE extension for VS Code. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for VS Code currently supports VS Code on macOS and Windows OS for Java applications.

Extension features

The extension has the following features:

- **Deploy applications directly from VS Code:**

  Rapidly iterate on your applications on Tanzu Application Platform by deploying them as workloads directly from within VS Code.

- **See code updates running on-cluster in seconds:**

  With Live Update (facilitated by Tilt), you can deploy your workload once, save changes to the code and then see those changes reflected within seconds in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies and other variables updated.

- **See workloads running on the cluster:**

  From the Tanzu Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Overview of Tanzu Developer Tools for VS Code

VMware Tanzu Developer Tools for Visual Studio Code (VS Code) is the official VMware Tanzu IDE extension for VS Code. The extension helps you develop with Tanzu Application Platform and enables you to rapidly iterate on your workloads on supported Kubernetes clusters that have Tanzu Application Platform installed.

Tanzu Developer Tools for VS Code currently supports VS Code on macOS and Windows OS for Java applications.

Extension features

The extension has the following features:

- **Deploy applications directly from VS Code:**

  Rapidly iterate on your applications on Tanzu Application Platform by deploying them as workloads directly from within VS Code.

- **See code updates running on-cluster in seconds:**

  With Live Update (facilitated by Tilt), you can deploy your workload once, save changes to the code and then see those changes reflected within seconds in the workload running on the cluster.

- **Debug workloads directly on the cluster:**

  Debug your application in a production-like environment by debugging on your Kubernetes cluster that has Tanzu Application Platform. An environment's similarity to production relies on keeping dependencies and other variables updated.

- **See workloads running on the cluster:**

  From the Tanzu Workloads panel you can see any workload found within the cluster and namespace specified in the current kubectl context.

Note

The new variation of Out of the Box (OOTB) Basic supply chains, which outputs Carvel packages to enable configuring multiple runtime environments, is not currently supported. For more information about the variation, see Carvel Package Supply Chains.

Install Tanzu Developer Tools for VS Code

This topic tells you how to install VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisites

Before installing the extension, you must have:

- VS Code
- kubectl
- Tilt v0.30.12 or later
- Tanzu CLI and plug-ins
- A cluster with the Tanzu Application Platform Full profile or Iterate profile

If you are an app developer, someone else in your organization might have already set up the Tanzu Application Platform environment.

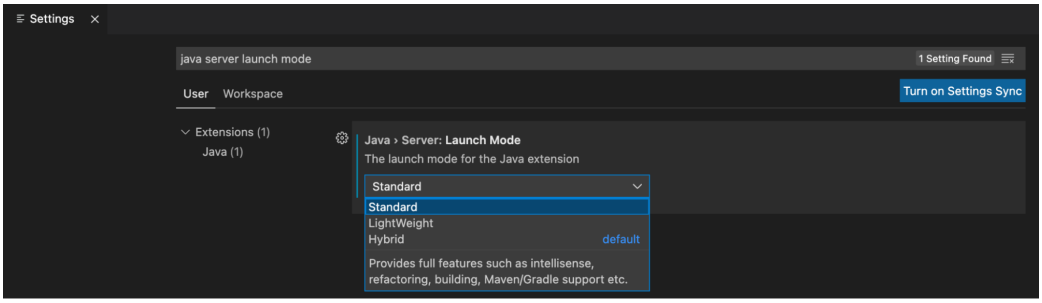
Docker Desktop and local Kubernetes are not prerequisites for using Tanzu Developer Tools for VS Code.

Install

To install the extension:

1. Sign in to VMware Tanzu Network and download Tanzu Developer Tools for Visual Studio Code.
2. Open VS Code.
3. Press cmd+shift+P to open the Command Palette and run `Extensions: Install from VSIX...`.

4. Select the extension file **tanzu-vscode-extension.vsix**.
5. If you do not have the following extensions, and they do not automatically install, install them from VS Code Marketplace:
    - Debugger for Java
    - Language Support for Java(™) by Red Hat
    - YAML
6. Ensure Language Support for Java is running in Standard Mode. You can configure it in the **Settings** menu by going to **Code** > **Preferences** > **Settings** under **Java > Server: Launch Mode**.



When the JDK and Language Support for Java are configured correctly, you see that the integrated development environment creates a directory target where the code is compiled.

Configure

To configure VMware Tanzu Developer Tools for VS Code:

1. Ensure that you are targeting the correct cluster. For more information, see the Kubernetes documentation.
2. Go to **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** and set the following:
    - **Confirm Delete**: This controls whether the extension asks for confirmation when deleting a workload.
    - **Enable Live Hover**: For more information, see Integrating Live Hover by using Spring Boot Tools. Reload VS Code for this change to take effect.

- **Source Image**: (Required) The registry location for publishing local source code. For example, `registry.io/yourapp-source`. This must include both a registry and a project name.
- **Local Path**: (Optional) The path on the local file system to a directory of source code to build. This is the current directory by default.
- **Namespace**: (Optional) This is the namespace that workloads are deployed into. The namespace set in `kubeconfig` is the default.

Uninstall

To uninstall VMware Tanzu Developer Tools for VS Code:

1. Go to **Code** > **Preferences** > **Settings** > **Extensions**.
2. Right-click the extension and select **Uninstall**.

Next steps

Proceed to Getting started with Tanzu Developer Tools for Visual Studio Code.

Get started with Tanzu Developer Tools for VS Code

This topic guides you through getting started with VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Prerequisite

Install VMware Tanzu Developer Tools for Visual Studio Code.

Configure source image registry

Before deploying a workload, you must authenticate with an image registry to store your source code. You can use the Docker CLI to authenticate or you can set environment variables that the Tanzu CLI can use to authenticate.

Docker CLI
To authenticate by using the Docker CLI, run:

```
docker login $REGISTRY_HOSTNAME -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
```

Tanzu CLI
To authenticate using the Tanzu CLI, export these environment variables by running:

```
export TANZU_APPS_REGISTRY_CA_CERT=PATH-TO-CA-CERT.nip.io.crt
export TANZU_APPS_REGISTRY_PASSWORD=USERNAME
export TANZU_APPS_REGISTRY_USERNAME=PASSWORD
```

`CA_CERT` is only needed for a custom or private registry.

For more information, see Workload creation fails due to authentication failure in Docker Registry

Set up Tanzu Developer Tools

The extension makes use of the following files within your project:

- `workload.yaml`
- `catalog-info.yaml`
- `Tiltfile`
- `.tanzuignore`

You can create these files by using the instructions in this topic, or use the files in the View an example project section.

There are two ways to create these files:

- Using the code snippets that Tanzu Developer Tools provide, which create templates in empty files that you then fill in with the required information.
- Writing the files manually.

Create the `workload.yaml` file

`workload.yaml` provides instructions to the Supply Chain Choreographer about how to build and manage a workload.

The extension requires only one `workload.yaml` file per project. `workload.yaml` must be a single-document YAML file, not a multidocument YAML file.

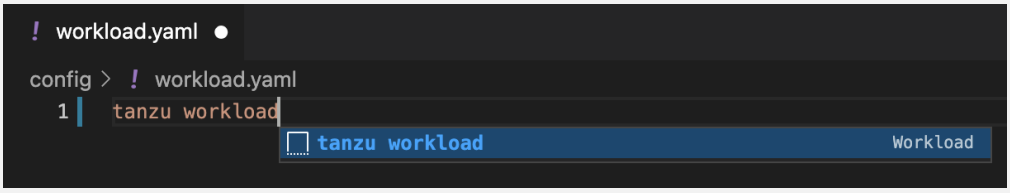
Before beginning to write your `workload.yaml` file, ensure that you know:

- The name of your application. For example, `my app`.
- The workload type of your application. For example, `web`.
- The GitHub source code URL. For example, `github.com/mycompany/myapp`.
- The Git branch of the source code that you intend to use. For example, `main`.

Code snippets

To create a `workload.yaml` file by using code snippets:

1. (Optional) Create a directory named `config` in the root directory of your project. For example, `my project/config`.
2. Create a file named `workload.yaml` in the new config directory. For example, `my project/config/workload.yaml`.
3. Open the new `workload.yaml` file in VS Code, enter `tanzu workload` in the file to trigger the code snippets, and either press Enter or left-click the `tanzu workload` text in the drop-down menu.



4. Fill in the template by pressing the Tab key.

Manual

To create your `workload.yaml` file manually, follow this example:

```
apiVersion: carto.run/v1alpa1
kind: Workload
metadata:
 name: APP-NAME
 labels:
   apps.tanzu.vmware.com/workload-type: WORKLOAD-TYPE
```

```
    app.kubernetes.io/part-of: APP-NAME
spec:
 source:
   git:
     url: GIT-SOURCE-URL
     ref:
       branch: GIT-BRANCH-NAME
```

Where:

- `APP-NAME` is the name of your application.
- `WORKLOAD-TYPE` is the type of this workload. For example, `web`.
- `GIT-SOURCE-URL` is your GitHub source code URL.
- `GIT-BRANCH-NAME` is the Git branch of your source code.

Alternatively, you can use the Tanzu CLI to create a `workload.yaml` file. For more information about the Tanzu CLI command, see Tanzu apps workload apply in the Tanzu CLI documentation.

Create the `catalog-info.yaml` file

`catalog-info.yaml` enables the workloads of this project to appear in Tanzu Application Platform GUI.

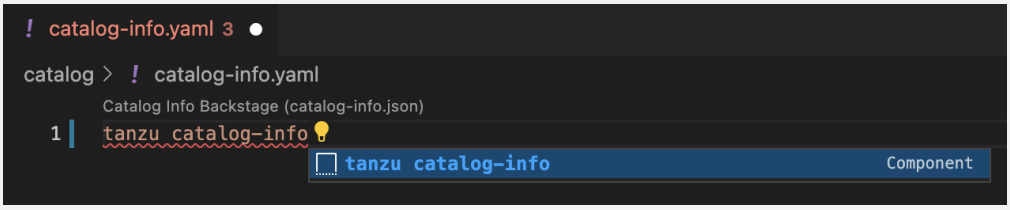
Before beginning to write your `catalog-info.yaml` file, ensure that you:

- Know the name of your application. For example, `my app`.
- Have a description of your application ready.

Code snippets

To create a `catalog-info.yaml` file by using the code snippets:

1. (Optional) Create a directory named `catalog` in the root directory of your project. For example, `my project/catalog`.
2. Create a file named `catalog-info.yaml` in the new config directory. For example, `my project/catalog/catalog-info.yaml`.
3. Open the new `catalog-info.yaml` file in VS Code, enter `tanzu catalog-info` in the file to trigger the code snippets, and then either press Enter or left-click the `tanzu catalog-info` text in the drop-down menu.



4. Fill in the template by pressing the Tab key.

Manual

To create your `catalog-info.yaml` file manually, follow this example:

```
apiVersion: backstage.io/v1alpha1
kind: Component
metadata:
 name: APP-NAME
 description: APP-DESCRIPTION
 tags:
   - tanzu
 annotations:
```

```
   'backstage.io/kubernetes-label-selector': 'app.kubernetes.io/part-of=APP-NAME'
spec:
 type: service
 lifecycle: experimental
 owner: default-team
```

Where:

- `APP-NAME` is the name of your application
- `APP-DESCRIPTION` is the description of your application

Create the Tiltfile file

The Tiltfile file provides the Tilt configuration to enable your project to Live Update on your Kubernetes cluster that has Tanzu Application Platform. The Tanzu Developer Tools extension requires only one **Tiltfile** per project.

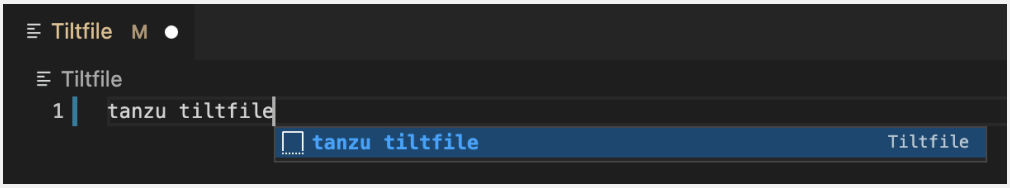
Before beginning to write your Tiltfile file, ensure that you know:

- The name of your application. For example, `my app`.
- The value of the source image. For example, `docker.io/mycompany/myapp`.
- Whether you want to compile the source image from a local directory other than the project directory or otherwise leave the `local path` value unchanged. For more information, see local path in the glossary.
- The path to your `workload.yaml` file. For example, `config/workload.yaml`.
- The name of your current Kubernetes context, if the targeting Kubernetes cluster enabled by Tanzu Application Platform is not running on your local machine.

Code Snippets

To create a Tiltfile file by using the code snippets:

1. Create a file named `Tiltfile` with no file extension in the root directory of your project. For example, `my project/Tiltfile`.
2. Open the new Tiltfile file in VS Code and enter `tanzu tiltfile` in the file to trigger the code snippets, and then either press Enter or left-click the `tanzu tiltfile` text in the drop-down menu.



3. Fill in the template by pressing the Tab key.
4. If the targeting Kubernetes cluster enabled by Tanzu Application Platform is not running on your local machine, add a new line to the end of the **Tiltfile** template and enter:

   ```
   allow_k8s_contexts('CONTEXT-NAME')
   ```

   Where `CONTEXT-NAME` is the name of your current Kubernetes context.

Manual

To create a Tiltfile file manually, follow this example:

```
SOURCE_IMAGE = os.getenv("SOURCE_IMAGE", default='SOURCE-IMAGE')
LOCAL_PATH = os.getenv("LOCAL_PATH", default='.')
```

```
NAMESPACE = os.getenv("NAMESPACE", default='default')

k8s_custom_deploy(
   'APP-NAME',
   apply_cmd="tanzu apps workload apply -f PATH-TO-WORKLOAD-YAML --live-update" +
       " --local-path " + LOCAL_PATH +
       " --SOURCE-IMAGE " + SOURCE_IMAGE +
       " --namespace " + NAMESPACE +
       " --yes >/dev/null" +
       " && kubectl get workload APP-NAME --namespace " + NAMESPACE + " -o yaml",
   delete_cmd="tanzu apps workload delete -f PATH-TO-WORKLOAD-YAML --namespace " + N
AMESPACE + " --yes" ,
   deps=['pom.xml', './target/classes'],
   container_selector='workload',
   live_update=[
       sync('./target/classes', '/workspace/BOOT-INF/classes')
   ]
)

k8s_resource('APP-NAME', port_forwards=["8080:8080"],
   extra_pod_selectors=[{'carto.run/workload-name': 'APP-NAME', 'app.kubernetes.io/c
omponent': 'run'}])
allow_k8s_contexts('CONTEXT-NAME')
```

Where:

- `SOURCE-IMAGE` is the value of source image.
- `APP-NAME` is the name of your application.
- `PATH-TO-WORKLOAD-YAML` is the local file system path to `workload.yaml`. For example, `config/workload.yaml`.
- `CONTEXT-NAME` is the name of your current Kubernetes context. If your Kubernetes cluster enabled by Tanzu Application Platform is running locally on your local machine, you can remove the entire `allow_k8s_contexts` line. For more information, see the Tilt documentation.

Create a `.tanzuignore` file

The `.tanzuignore` file specifies the file paths to exclude from the source code image. When working with local source code, you can exclude files from the source code to be uploaded within the image. Directories must not end with the system path separator (`/` or `\`). See this example. in GitHub.

View an example project

Before you begin, you need a container registry for the sample application.

You can view a sample application that demonstrates the necessary configuration files. There are two ways to obtain the sample application:

Application Accelerator
If your company has configured Application Accelerator, you can obtain the sample application there if it was not removed. To do so:

1. Open Application Accelerator.
2. Search for `Tanzu Java Web App` in Application Accelerator.
3. Add the required configuration information and generate the application.
4. Unzip the file and open the project in a VS Code workspace.

Clone from GitHub

To clone the sample application from GitHub:

1. Run `git clone` to clone the tanzu-java-web-app repository from GitHub.
2. Change into the `tanzu-java-web-app` directory.
3. Open the Tiltfile and replace `your-registry.io/project` with your container registry.

Next steps

Use Tanzu Developer Tools for VS Code.

Use Tanzu Developer Tools for VS Code

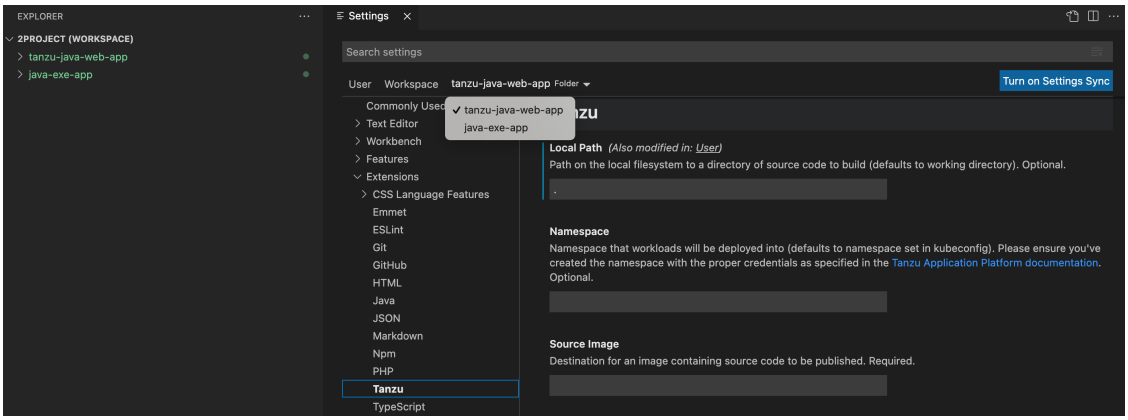
This topic tells you how to use VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Ensure that the project you want to use the extension with has the required files specified in Get started with Tanzu Developer Tools for VS Code.

The extension requires only one Tiltfile and one `workload.yaml` per project. The `workload.yaml` must be a single-document YAML file, not a multidocument YAML file.

Configure for multiple projects in the workspace

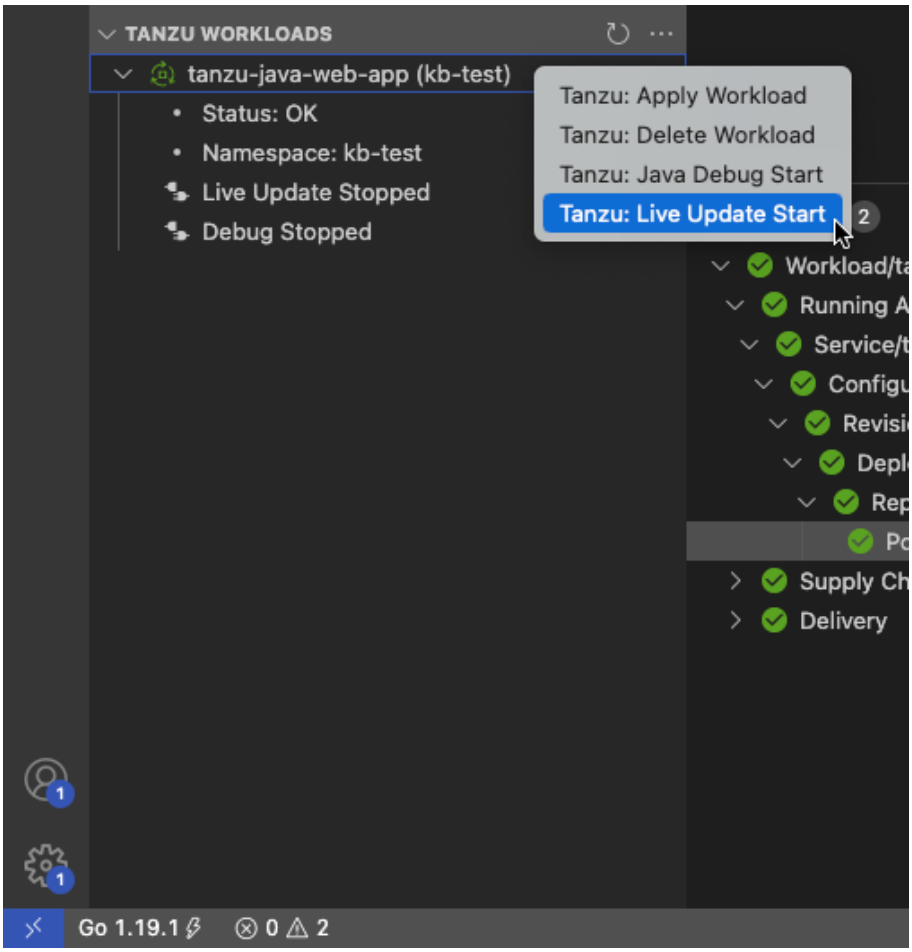
When working with multiple projects in a single workspace, you can configure the extension settings on a per-project basis by using the drop-down menu in **Settings**.



Workload Commands

All commands are available by right-clicking anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or in the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on Windows).

- Screenshot of pop-up menu opened from the workload panel:



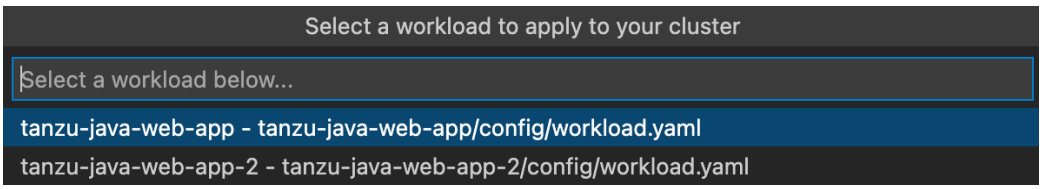
- Screenshot of the command palette:

Apply a workload

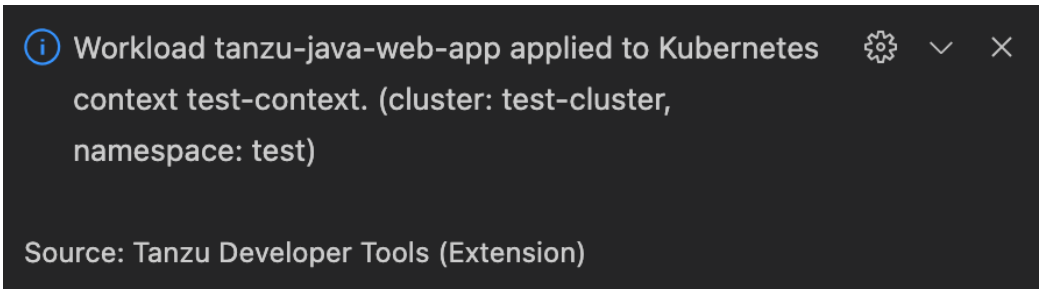
The extension enables you to apply workloads on your Kubernetes cluster that has Tanzu Application Platform.

To apply a workload:

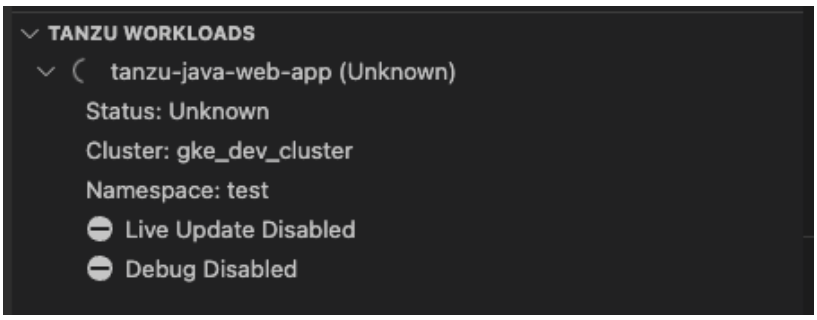
1. Right-click anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or open the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on Windows).
2. Select the `Tanzu: Apply Workload` command.
3. If applicable, select the workload to apply.

   A notification appears showing that the workload was applied.



A new workload appears on the Tanzu Workloads panel.



After the workload is deployed, the status on the Tanzu Workloads panel changes to `Ready`.

Debugging on the cluster

The extension enables you to debug your application on your Kubernetes cluster that has Tanzu Application Platform.

Debugging requires a `workload.yaml` file in your project. For information about creating a `workload.yaml` file, see Get Started with Tanzu Developer Tools for VS Code.

The developer sandbox experience enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

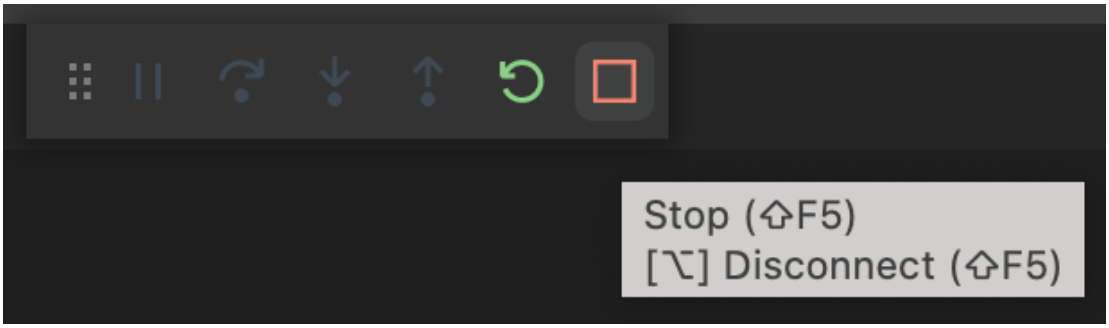
Start debugging on the cluster

To start debugging on the cluster:

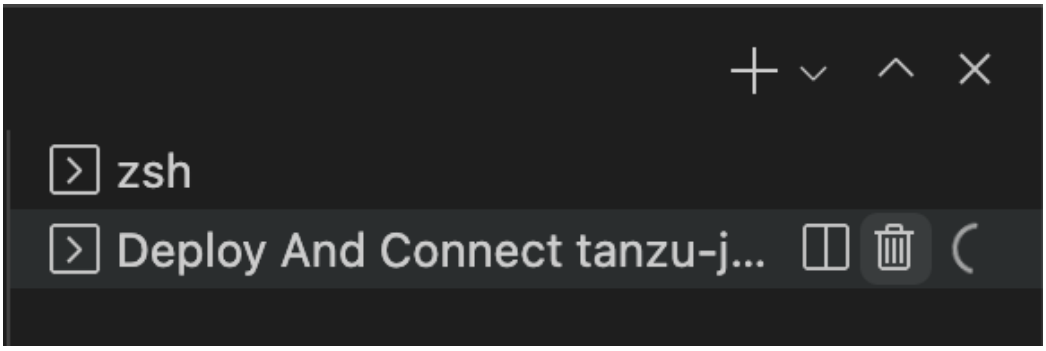
1. Add a breakpoint in your code.
2. Right-click anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or open the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on Windows).
3. Select the `Tanzu: Java Debug Start` command..

Stop Debugging on the cluster

To stop debugging on the cluster, you can click the stop button in the Debug overlay.



Alternatively, you can press ⌘+J (Ctrl+J on Windows) to open the panel and then click the trash can button for the debug task running in the panel.



Debug apps in a microservice repository

To debug multiple apps in a microservice repository:

1. Add each app folder as a workspace folder. For instructions, see the Visual Studio Code documentation.
2. Update the `tanzu.debugPort` setting so that it does not conflict with other debugging sessions. For how to update individual workspace folder settings, see the Visual Studio Code documentation.

Live Update

With the use of Live Update facilitated by Tilt, the extension enables you to deploy your workload once, save changes to the code, and see those changes reflected in the workload running on the cluster within seconds.

Live Update requires a `workload.yaml` file and a Tiltfile in your project. For information about how to create a `workload.yaml` and a Tiltfile, see Get Started with Tanzu Developer Tools for VS Code.

The developer sandbox experience enables developers to Live Update their code, and simultaneously debug the updated code, without having to deactivate Live Update when debugging.

Start Live Update

Before using Live Update, verify that your auto-save setting is either off or on with a delay. The delay must be long enough for the application to restart between auto saves to allow enough time for your app to Live Update when files change. The auto-save setting is in **Preferences** > **Text Editor** > **Files** > **Auto Save** > **Auto Save Delay**.

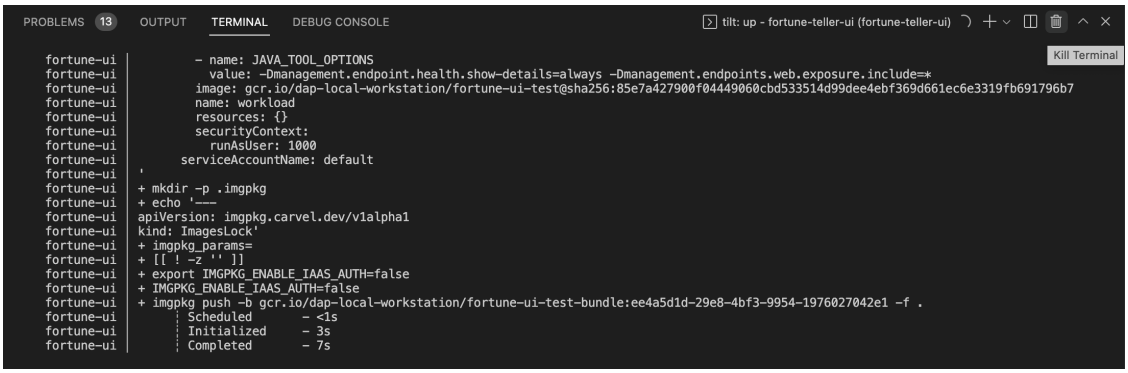
To start Live Update:

1. Right-click anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or open the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on Windows).
2. Select the `Tanzu: Live Update Start` command.

Stop Live Update

When Live Update stops, your application continues to run on the cluster, but the changes you made and saved in your editor are not present in your running application unless you redeploy your application to the cluster.

To stop Live Update, click the trash can button in the terminal pane to stop the Live Update process.



Deactivate Live Update

You can remove the Live Update capability from your application entirely. You might find this option useful in a troubleshooting scenario. Deactivating Live Update redeploys your workload to the cluster and removes the Live Update capability.

To deactivate Live Update:

1. Press ⇧⌘P (Ctrl+Shift+P on Windows) to open the Command Palette.
2. Run `Tanzu: Live Update Disable`.

3. Type the name of the workload for which you want to deactivate Live Update.

Live Update status

The current status of Live Update is visible on the right side of the status bar at the bottom of the VS Code window.

The Live Update status bar entry shows the following states:

- Live Update Stopped
- Live Update Starting...
- Live Update Running

To hide the Live Update status bar entry, right-click it and then click **Hide 'Tanzu Developer Tools (Extension)'**.

Live Update apps in a microservices repository

To Live Update multiple apps in a microservice repository:

1. Add each app folder as a workspace folder. For instructions, see the Visual Studio Code documentation.

2. Ensure that a port is available to port-forward the Knative service. For example, you might have this in your Tiltfile:

```
k8s_resource('tanzu-java-web-app', port_forwards=["NUMBER:8080"],
             extra_pod_selectors=[{'carto.run/workload-name': 'tanzu-java-web-ap
p', 'app.kubernetes.io/component': 'run'}])
```

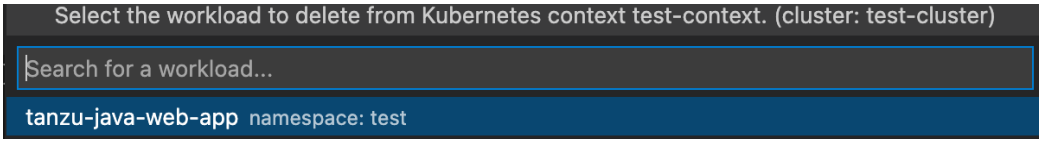
Where `NUMBER` is the port you choose. For example, `port_forwards=["9999:8080"]`.

Delete a workload

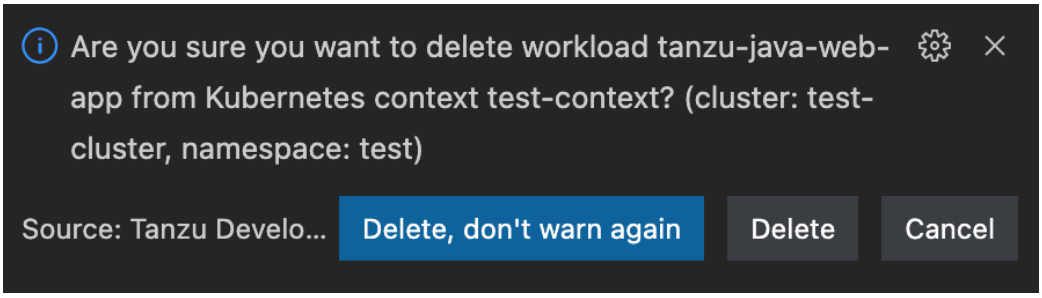
The extension enables you to delete workloads on your Kubernetes cluster that has Tanzu Application Platform.

To delete a workload:

1. Right-click anywhere in the VS Code project explorer, on an associated workload in the Tanzu Workloads panel, or open the Command Palette (⇧⌘P on Mac and Ctrl+Shift+P on Windows).
2. Select the `Tanzu: Delete Workload` command.
3. If applicable, select the workload to delete.



If the **Tanzu: Confirm Delete** setting is enabled, a message appears that prompts you to delete the workload and not warn again, delete the workload, or cancel.

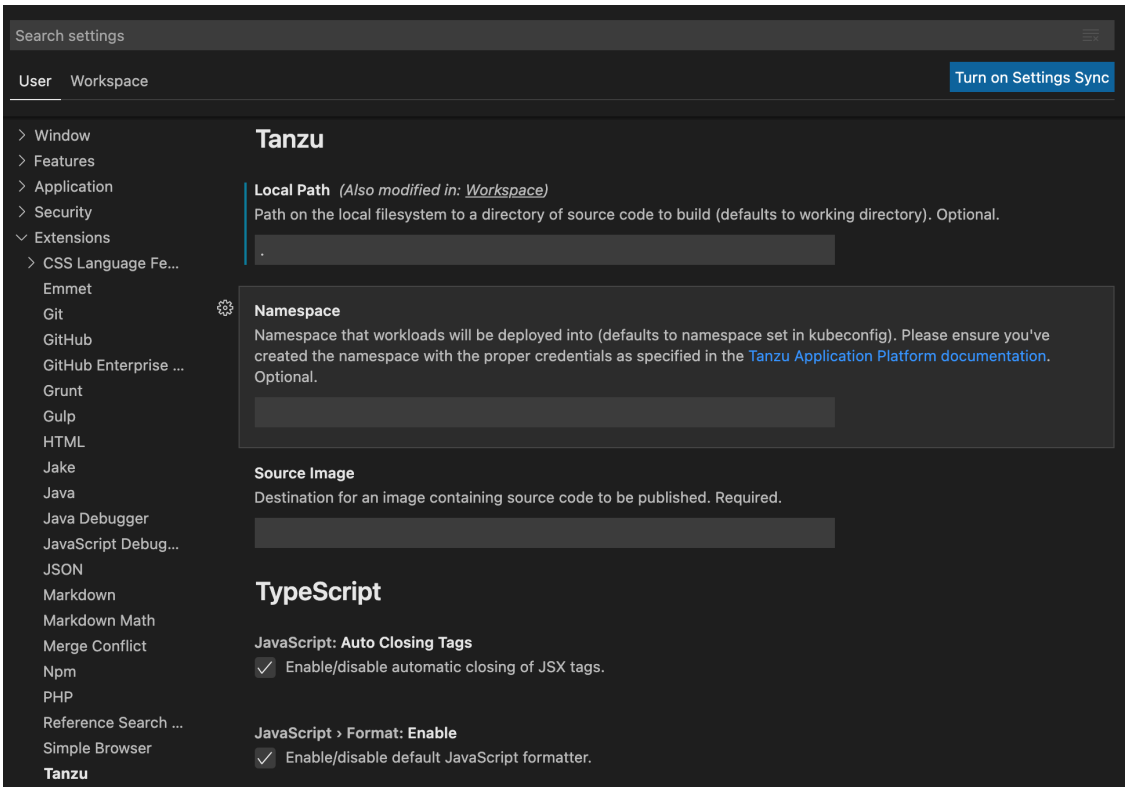


A notification appears showing that the workload was deleted.

Switch namespaces

To switch the namespace where you created the workload:

1. Go to **Code** > **Preferences** > **Settings**.
2. Expand the **Extensions** section of the settings and click **Tanzu**.
3. In the **Namespace** option, add the namespace you want to deploy to. This is the `default` namespace by default.



Tanzu Workloads panel

The current state of the workloads is visible in the Tanzu Workloads view. This view is a separate section in the bottom of the Explorer view in the Side Bar. The view shows the current status of each workload, namespace, and cluster. It also shows whether Live Update and Debug is running, stopped, or deactivated.

The Tanzu Activity tab in the Panels view enables developers to visualize the supply chain, delivery, and running application pods. The tab enables a developer to view and describe logs on each resource associated with a workload from within their IDE. The tab displays detailed error messages for each resource in an error state.

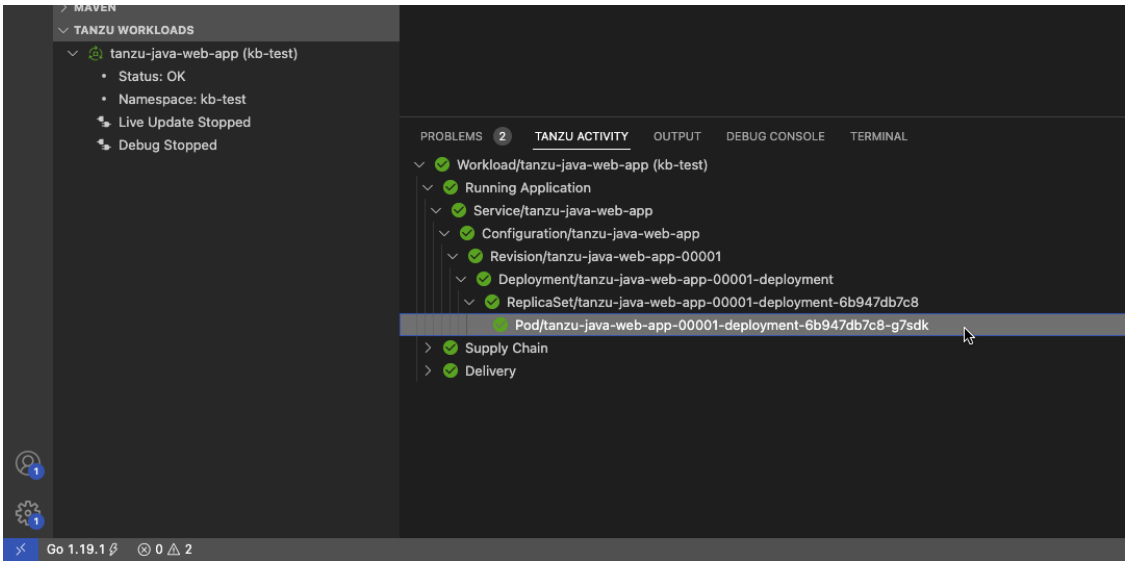
Workload commands are available from the Tanzu Workloads panel on workloads that have an associated module in the current project.

This association is based on a module name and a workload name matching. For example, a project with a module named `my-app` is associated with a deployed workload named `my-app`.

When taking an action from the Tanzu Workloads panel, the action uses the namespace of the deployed workload regardless of the configuration in the module.

For example, you might have a Live Update configuration with a namespace argument of `my-apps-1`, but running the action from a deployed workload in namespace `my-apps-2` starts a Live Update session with a namespace argument of `my-apps-2`.

The Tanzu Workloads panel uses the cluster and defaults to the namespace specified in the current kubectl context.



To add a namespace:

1. View the current context and namespace by running:

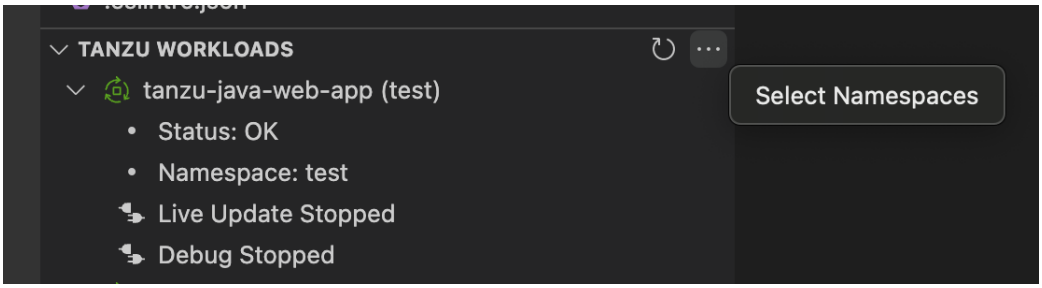
   ```
   kubectl config get-contexts
   ```

2. Set a namespace for the current context by running:

   ```
   kubectl config set-context --current --namespace=YOUR-NAMESPACE
   ```

3. Use one of these methods to add additional namespaces to your Tanzu Workloads panel:

   - Go to **Preferences** > **Extensions** > **Tanzu Developer Tools** > **Tracked Namespaces** and then select the namespaces that you want.
   - Go to **Workload Panel** > **Additional Options** > **Select Namespaces** and then select the namespaces that you want.



Working with Microservices in a Monorepo

A monorepo is single Git repository that contains multiple workloads. Each individual workload is placed in a subfolder of the main repository.

You can find an example of this in Application Accelerator. The relevant accelerator is called Spring SMTP Gateway, and you can obtain its source code as an accelerator or directly from the application-accelerator-samples GitHub repository.

This project exemplifies a typical layout:

- `MONO-REPO-ROOT/`
  - `pom.xml` (parent pom)
  - `microservice-app-1/`

- `pom.xml`
- `mvnw` (and other mvn-related files for building the workload)
- `Tiltfile` (supports Live Update)
- `config`
  - `workload.yaml` (supports deploying and debugging from IntelliJ)
- `src/` (contains source code for this microservice)
- `microservice-app-2/`
- …similar layout

Recommended structure: Microservices that can be built independently

In this example, each of the microservices can be built independently of one another. Each subfolder contains everything needed to build that workload.

This is reflected in the `source` section of `workload.yaml` by using the `subPath` attribute:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
  name: microservice-app-1
  ...
spec:
  source:
    git:
      ref:
        branch: main
      url: https://github.com/kdvolder/sample-mono-repo.git
    subPath: microservice-app-1 # build only this
  ...
```

For setting up your own repositories, it's best practice to set up a monorepo so that each microservice can be built completely independently.

To work with these monorepos:

- Import the monorepo as a project into VS Code.
- Interact with each of the subfolders in the same way you would interact with a project containing a single workload.

Alternative structure: Services with build-time interdependencies

Some monorepos do not have submodules that can be independently built. Instead the `pom.xml` files of the submodules are set up to have some build-time interdependencies. For example:

- A submodule `pom.xml` can reference the parent `pom.xml` as a common place for centralized dependency management.
- A microservice submodule can reference another, as a maven dependency.
- Several microservice submodules can reference one or more shared library modules.

For these projects, make these adjustments:

1. Make `workload.yaml` point to the repository root, not a subfolder. Because submodules have dependencies on code outside of their own subfolder, all source code from the

repository must be supplied to the workload builder.

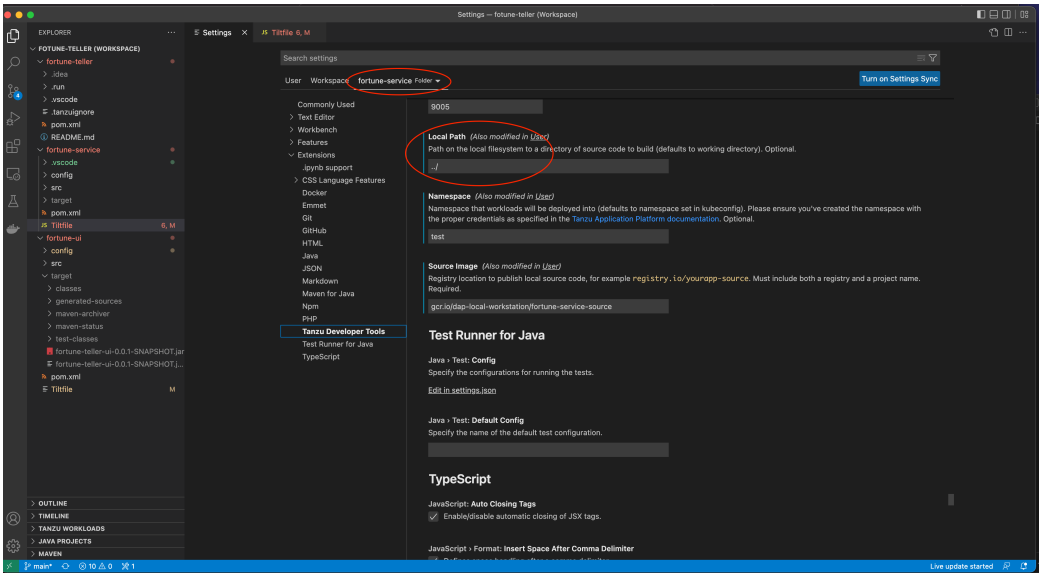
2. Make `workload.yaml` specify additional buildpack arguments through environment variables. They differentiate the submodule that the build is targeting.

   Both of these `workload.yaml` changes are in the following example:

```
apiVersion: carto.run/v1alpha1
kind: Workload
metadata:
name: fortune-ui
labels:
  apps.tanzu.vmware.com/workload-type: web
  app.kubernetes.io/part-of: fortune-ui
spec:
build:
  env:
    - name: BP_MAVEN_BUILD_ARGUMENTS
      value: package -pl fortune-teller-ui -am # indicate which module to build.
    - name: BP_MAVEN_BUILT_MODULE
      value: fortune-teller-ui # indicate where to find the built artefact to de
ploy.
source:
  git:
    url: https://github.com/my-user/fortune-teller # repo root
    ref:
      branch: main
```

   For more information about these and other `BP_xxx` buildpack parameters, see the Buildpack Documentation.

3. Make the local path preference for each subfolder point to the path of the repository root Because submodules have dependencies on code outside of their own subfolder, all source code from the repository must be supplied to the workload builder.



Changing logging verbosity

The Tanzu Language Server saves logs to `~/tanzu-langserver.log`. You can change the log verbosity in **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools** > **Language Server: Log Verbosity**.

Pinniped compatibility

This topic tells you the compatibility details of Pinniped in GitHub.

OAuth

OAuth login is compatible only when both `--skip-browser` and `--skip-listen` flags are not set.

LDAP

LDAP authentication is not compatible with VMware Tanzu Developer Tools for Visual Studio Code.

Integrate Live Hover by using Spring Boot Tools

For more information about this feature, see the **Live application information hovers** section of the Spring Boot Tools Marketplace page.

Prerequisites

To integrate Live Hover by using Spring Boot Tools you need:

- A Tanzu Spring Boot application, such as tanzu-java-web-app
- Spring Boot Extension Pack (includes Spring Boot Dashboard) extension

Activate the Live Hover feature

Activate the Live Hover feature by enabling it in **Code** > **Preferences** > **Settings** > **Extensions** > **Tanzu Developer Tools**.

Deploy a Workload to the Cluster

Follow these steps to deploy the workload for an app to a cluster, making live hovers appear. The examples in some steps reference the sample tanzu-java-web-app.

1. Clone the repository by running:

   ```
   git clone REPOSITORY-ADDRESS
   ```

   Where `REPOSITORY-ADDRESS` is your repository address. For example, `https://github.com/vmware-tanzu/application-accelerator-samples`.

2. Open the project in VS Code, with the Live Hover feature enabled, by running:

   ```
   TAP_LIVE_HOVER=true code ./PROJECT-DIRECTORY
   ```

   Where `PROJECT-DIRECTORY` is your project directory. For example, `./application-accelerator-samples/tanzu-java-web-app`.

3. Verify that you are targeting the cluster on which you want to run the workload by running:

   ```
   kubectl cluster-info
   ```

   For example:

   ```
   $ kubectl cluster-info
   Kubernetes control plane is running at https://...
   CoreDNS is running at https://...
   ```

```
To further debug and diagnose cluster problems, use 'kubectl cluster-info dum
p'.
```

Tanzu Developer Tools for VS Code periodically connects to your cluster to search for pods from which live data can be extracted and shown. Tanzu Developer Tools for VS Code uses your current context from `~/.kube/config` to choose which cluster to connect with.

4. If you don't have the workload running yet, run `Tanzu: Apply Workload` from the Command Palette. Tanzu Developer Tools for VS Code periodically searches for pods in your cluster that correspond to the workload configurations it finds in your workspace.

5. The workload takes time to build and then start a running pod. To see if a pod has started running, run:

   ```
   kubectl get pods
   ```

   For example:

   ```
   $ kubectl get pods
   NAME                                                READY   STATUS      REST
   ARTS   AGE
   tanzu-java-web-app-00001-deployment-8596bfd9b4-5vgx2   2/2     Running     0
   20s
   tanzu-java-web-app-build-1-build-pod                0/1     Completed   0
   2m26s
   tanzu-java-web-app-config-writer-fpnzb-pod          0/1     Completed   0
   67s
   ```

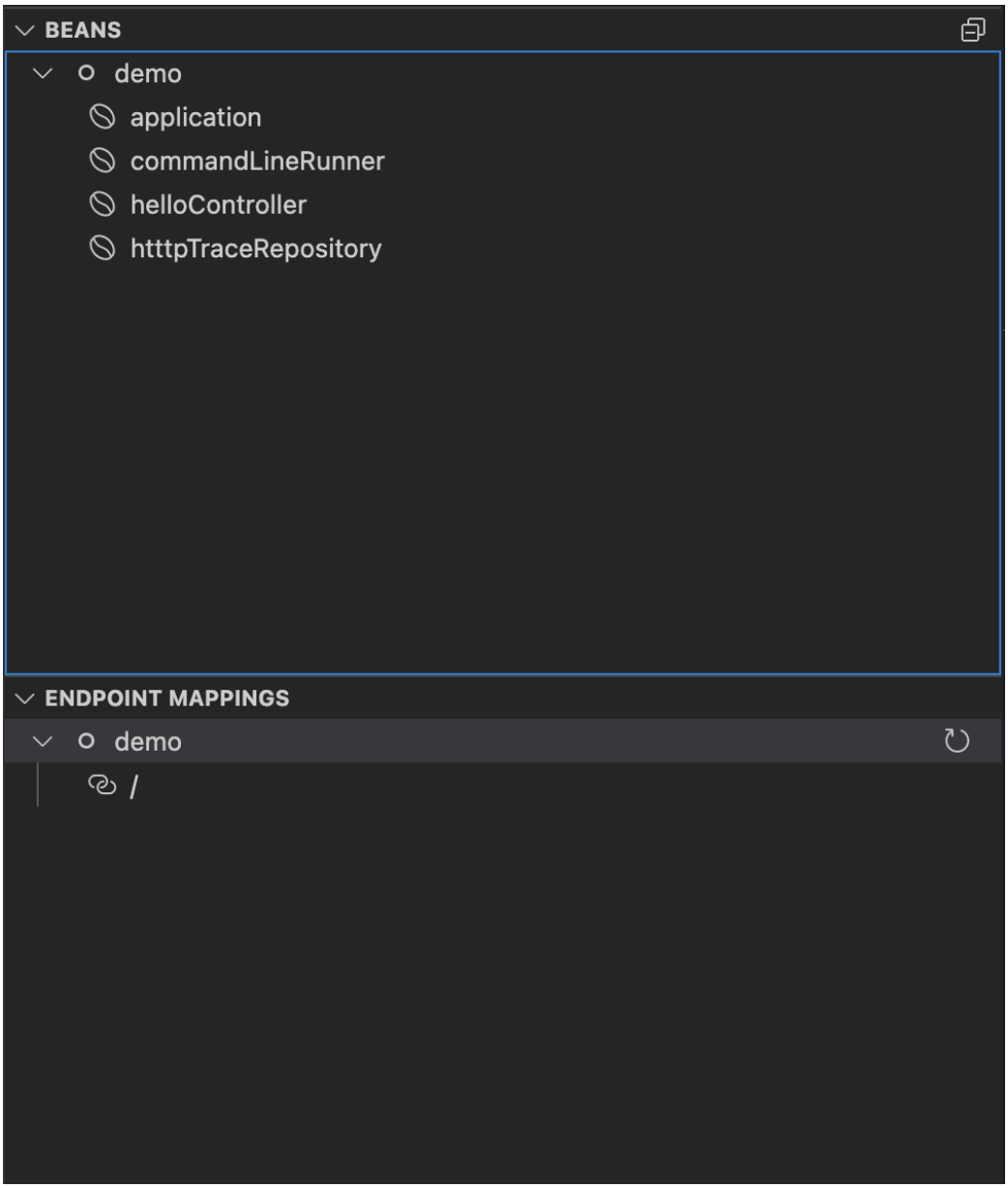
   In this example, live data can be extracted from the `...-0001-deployment-...` pod.

6. Open a Java file, such as `HelloController.java`. After a delay of up to 30 seconds, because of a 30-second polling loop, green highlights appear in your code.

7. Hover over any of the bubbles to see live information about the corresponding element.

8. The `Live Beans` and `Live Endpoint Mapping` information are displayed in Spring Boot Dashboard. To view the Spring Boot Dashboard, run `View: Show Spring Boot Dashboard` from the Command Palette.



Use Memory View in Spring Boot Dashboard

This topic tells you how to use Spring Boot Dashboard to view memory use.

For more information about Spring Boot Dashboard, see Spring Boot Dashboard.

Prerequisites

To see the Memory View in Spring Boot Dashboard you need:

- A Tanzu Spring Boot application, such as tanzu-java-web-app
- The Spring Boot Extension Pack, which includes Spring Boot Dashboard

Deploy a workload

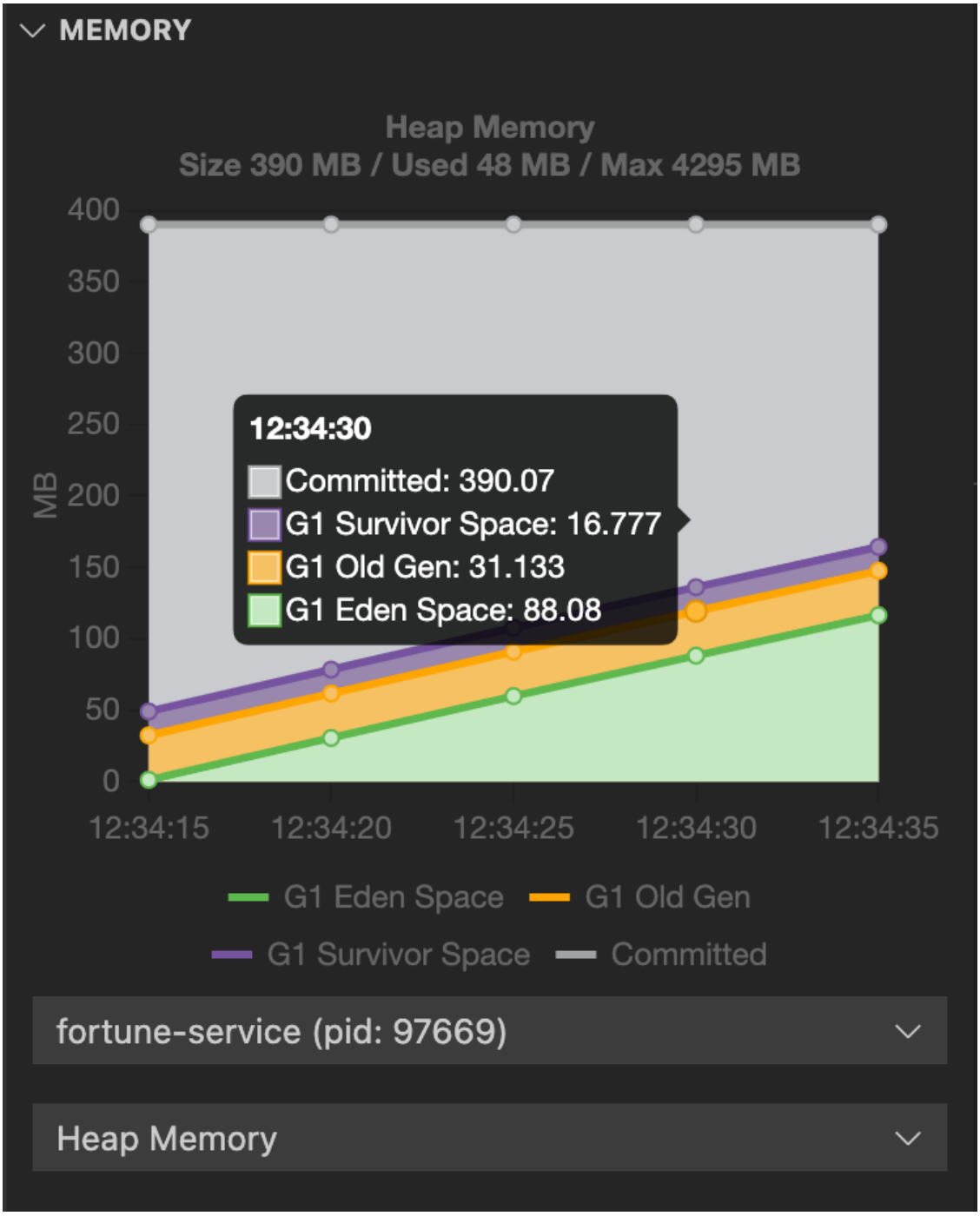
Deploy the workload for an app to a cluster by following the steps in Deploy a Workload to the Cluster.

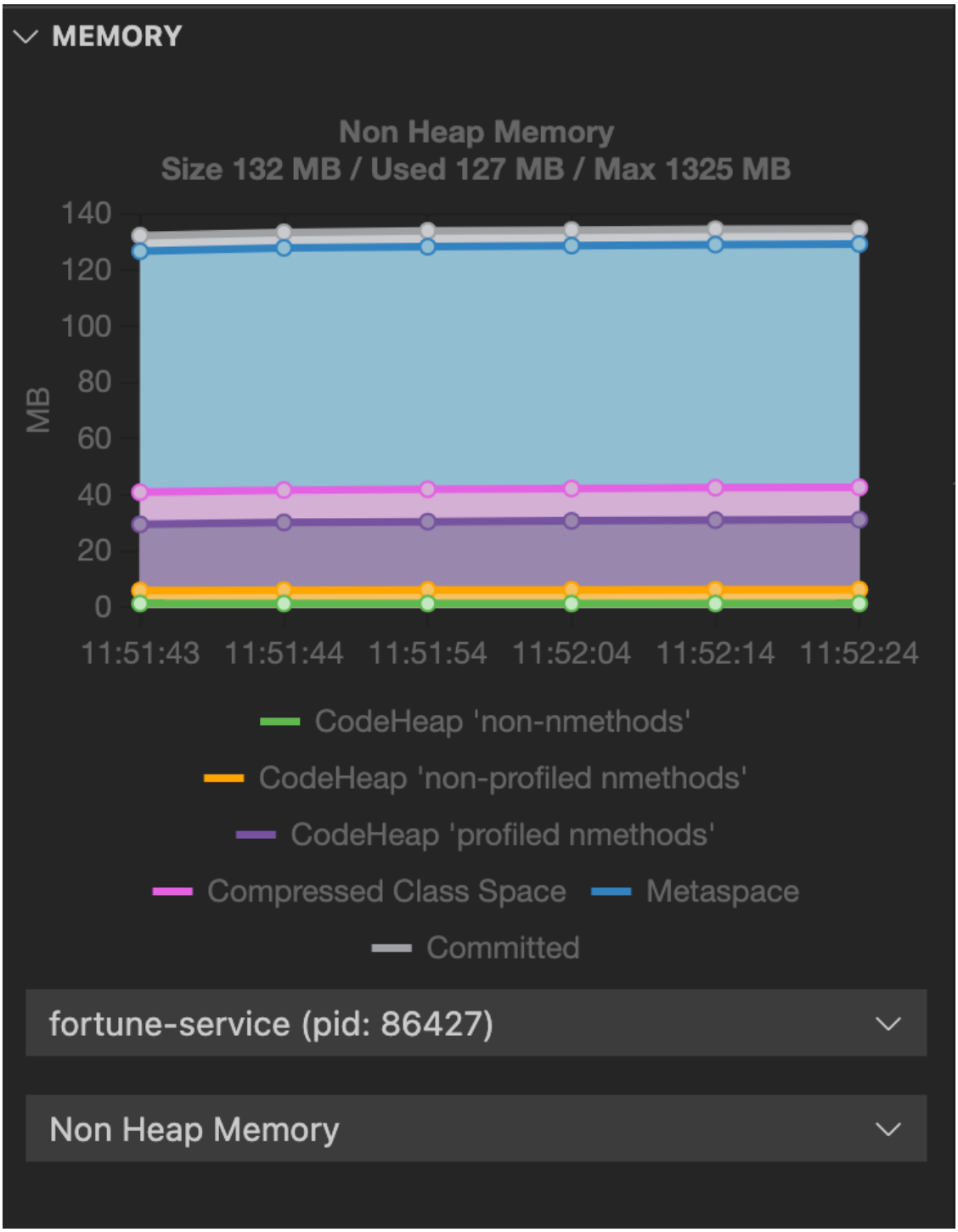
View memory use in Spring Boot Dashboard

To view the Spring Boot Dashboard, run `View: Show Spring Boot Dashboard` from the Command Palette.

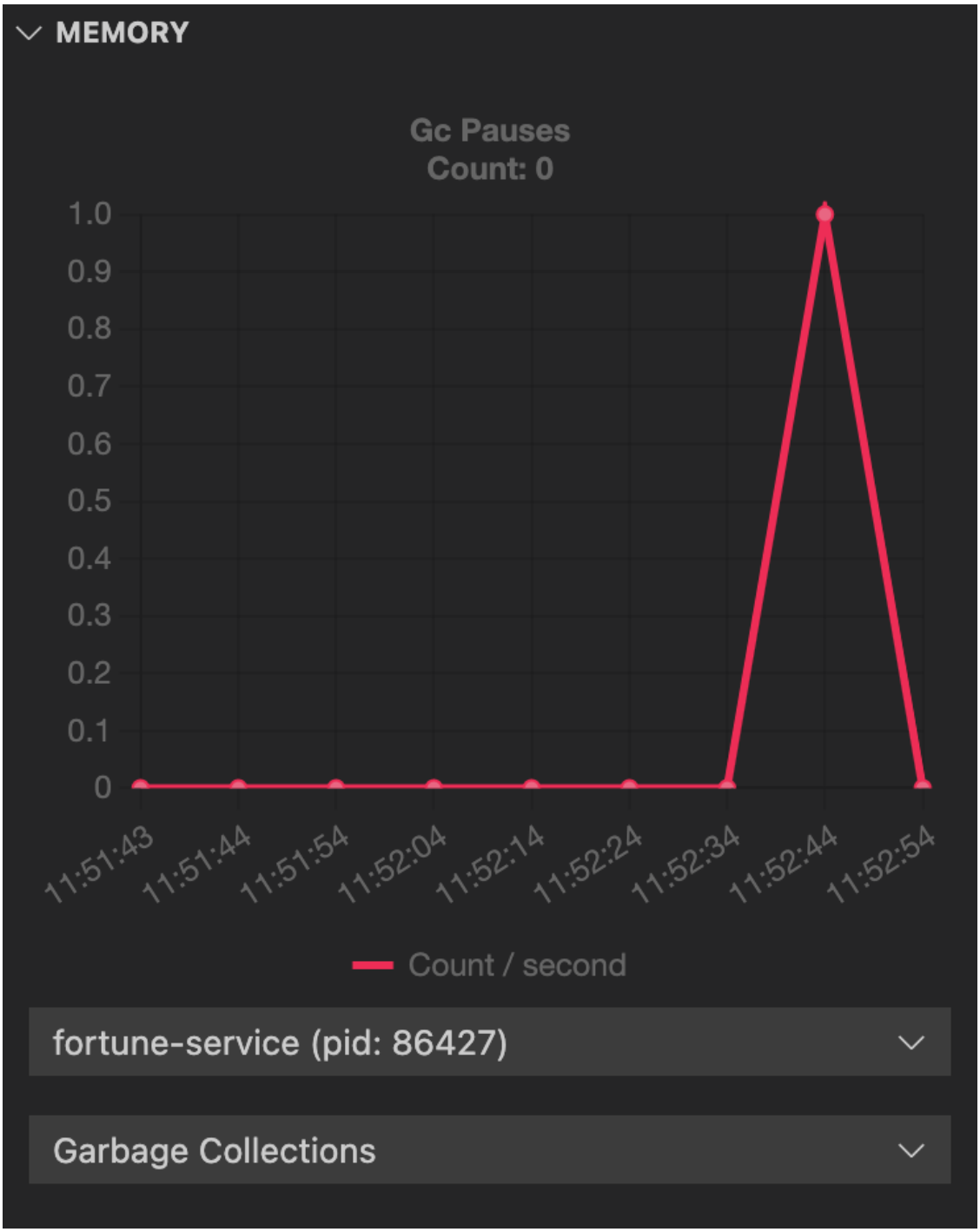
When the app is running, the Memory View section is displayed in Spring Boot Dashboard. The graphical representation in the memory view highlights the memory use inside the Java virtual machine (JVM). The drop-down menus beneath the graph enable you to switch between different running processes and graphical views.

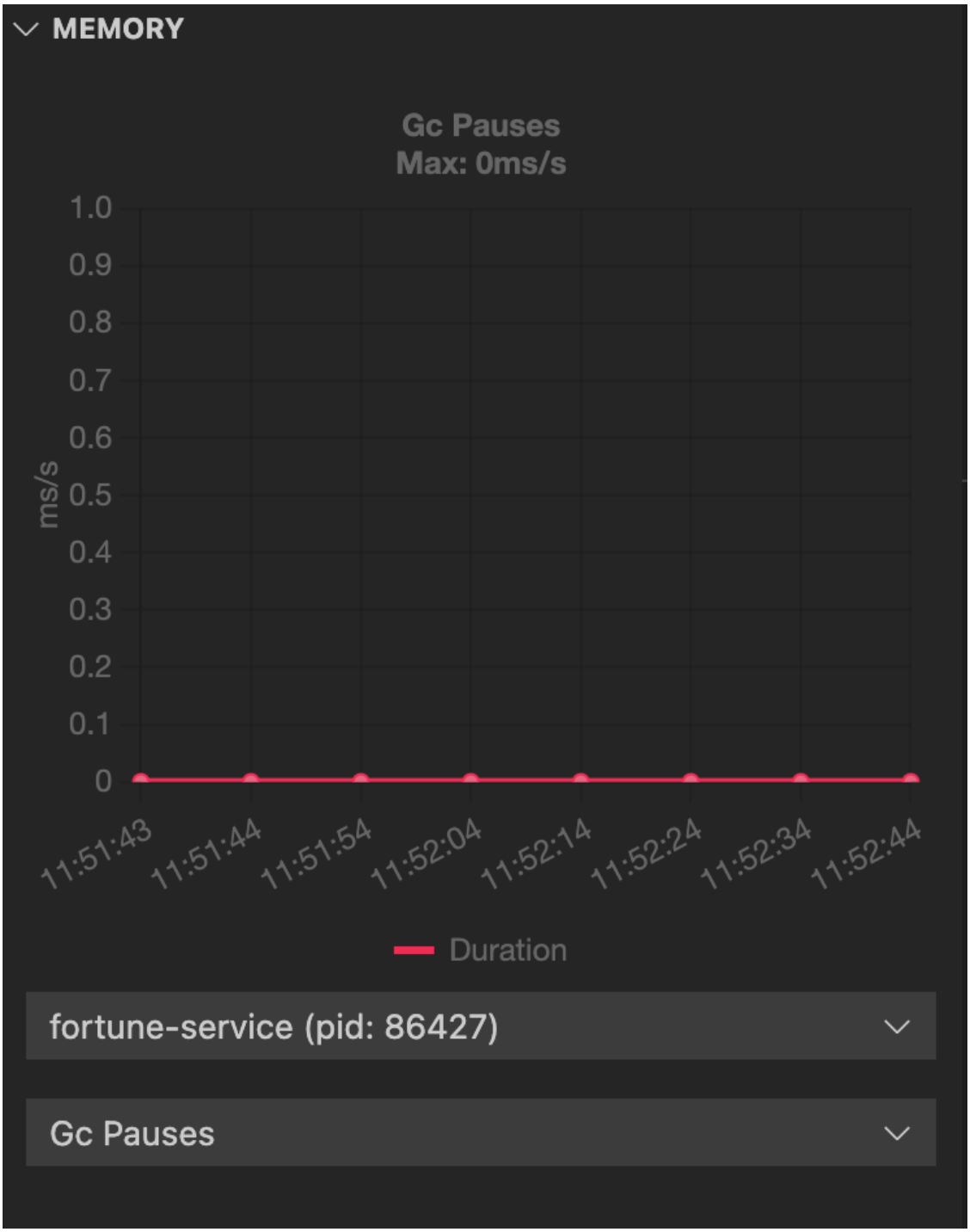
The heap and non-heap memory regions provide memory insights into the application. The real-time graphs display a stacked overview of the different spaces in memory relative to the total memory used and total memory size.



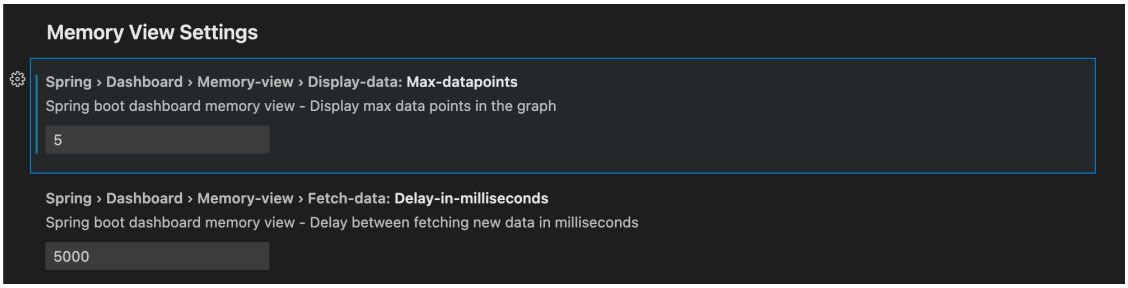


The memory view also contains graphs to display the garbage-collection pauses and garbage-collection events. Long and frequent garbage-collection pauses indicate that the app is having a memory problem that requires further investigation.


MEMORY
Gc Pauses
Count: 0
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
11:51:43
11:51:44
11:51:54
11:52:04
11:52:14
11:52:24
11:52:34
11:52:44
11:52:54
Count / second
fortune-service (pid: 86427)
Garbage Collections



The graphs show only real-time data. You can configure the number of data points to view and the interval by changing the settings. To access the settings on macOS, go to **Code** > **Preferences** > **Settings** > **Extensions** > **Spring Boot Dashboard** > **Memory View Settings**. The navigation path might differ on other operating systems, such as Windows and Linux.



Troubleshoot Tanzu Developer Tools for VS Code

This topic tells you what to do when you encounter issues with VMware Tanzu Developer Tools for Visual Studio Code (VS Code).

Unable to view workloads on the panel when connected to GKE cluster

Symptom

When connecting to Google's GKE clusters, an error appears with the text `WARNING: the gcp auth plugin is deprecated in v1.22+, unavailable in v1.25+; use gcloud instead.`

Cause

GKE authentication was extracted into a separate plug-in and is no longer inside the Kubernetes client or libraries.

Solution

Download and configure the GKE authentication plug-in. For instructions, see the Google documentation.

Live Update fails with `UnsupportedClassVersionError`

Symptom

After live-update has synchronized changes you made locally to the running workload, the workload pods start failing with an error message similar to the following:

```
Caused by: org.springframework.beans.factory.CannotLoadBeanClassException: Error loadi
ng class
[com.example.springboot.HelloController] for bean with name 'helloController' defined
in file
[/workspace/BOOT-INF/classes/com/example/springboot/HelloController.class]: problem wi
th class file
or dependent class; nested exception is
java.lang.UnsupportedClassVersionError: com/example/springboot/HelloController has bee
n compiled by
a more recent version of the Java Runtime (class file version 61.0), this version of t
he
Java Runtime only recognizes class file versions up to 55.0
```

Cause

The classes produced locally on your machine are compiled to target a later Java virtual machine (JVM). The error message mentions `class file version 61.0`, which corresponds to Java 17. The buildpack, however, is set up to run the application with an earlier JVM. The error message mentions `class file versions up to 55.0`, which corresponds to Java 11.

The root cause of this is a misconfiguration of the Java compiler that VS Code uses. The cause might be a suspected issue with the VS Code Java tooling, which sometimes fails to properly configure the compiler source and target compatibility-level from information in the Maven POM.

For example, in the `tanzu-java-web-app` sample application the POM contains the following:

```
<properties>
        <java.version>11</java.version>
```

```
        ...
</properties>
```

This correctly specifies that the app must be compiled for Java 11 compatibility. However, the VS Code Java tooling sometimes fails to take this information into account.

Solution

Force the VS Code Java tooling to re-read and synchronize information from the POM:

1. Right-click the `pom.xml` file.
2. Click **Reload Projects**.

This causes the internal compiler level to be set correctly based on the information from `pom.xml`. For example, Java 11 in `tanzu-java-web-app`.

Timeout error when Live Updating

Symptom

When you attempt to Live Update your workload, the following error message appears in the log:

```
ERROR: Build Failed: apply command timed out after 30s - see }}{{https://docs.tilt.dev/api.html#api.update_settings{{ for how to increase}}
```

Cause

Kubernetes times out on upserts over 30 seconds.

Solution

Add `update_settings (k8s_upsert_timeout_secs = 300)` to the Tiltfile. For more information, see the Tiltfile documentation.

Task-related error when running a Tanzu Debug launch configuration

Symptom

When you attempt to run a Tanzu Debug launch configuration, you see a task-related error message similar to the following:

```
Could not find the task 'tanzuManagement: Kill Port Forward my-app
```

Cause

The task you're trying to run is no longer supported.

Solution

Delete the launch configuration from your `launch.json` file in your `.vscode` directory.

Tanzu Workloads panel workloads only show delete command

Symptom

Some or all workloads in the Tanzu Workloads panel only have describe and delete actions.

Cause

By design, only associated workloads have apply, debug, and Live Update workload actions available.

Solution

Open a project that contains a module that can be associated with your deployed workloads.

Workload actions do not work when in a project with spaces in the name

Symptom

Workload actions do not work. The console displays an error message similar to the following:

```
Error: unknown command "projects/my-app" for "apps workload apply"Process finished wit
h exit code 1
```

Cause

On Windows, workload actions do not work when in a project with spaces in the name, such as `my-app project`.

Solution

1. Close the code editor.
2. Move or rename your project folder on the disk, ensuring that no part of its path contains any spaces.
3. Delete the project settings folder from the project to start with a clean slate. The folder is `.idea` if using IntelliJ and `.vscode` if using VS Code.
4. Open the code editor and then open the project in its new location.

Cannot apply workload because of a malformed kubeconfig file

Symptom

You cannot apply a workload. You see an error message when you attempt to do so.

Cause

Your kubeconfig file (`~/.kube/config`) is malformed.

Solution

Fix your kubeconfig file.

`config-writer-pull-requester` is categorized as Unknown

Symptom

In the Tanzu Activity Panel, the `config-writer-pull-requester` of type `Runnable` is incorrectly categorized as **Unknown**. It should be in the **Supply Chain** category.

Solution

A fix is planned for a future release.

Frequent application restarts

Symptom

When an application is applied from VS Code it restarts frequently.

Cause

An application or environment behavior is triggering the application to restart.

Observed trigger behaviors include:

- The application itself writing logs to the file system in the application directory that Live Update is watching
- Autosave being set to a very high frequency in the IDE configuration

Solution

Prevent the trigger behavior. Example solutions include:

- Prevent 12-factor applications from writing to the file system.
- Reduce the autosave frequency to once every few minutes.

Overview of Tekton

Tekton is a cloud-native, open-source framework for creating CI/CD systems. It allows developers to build, test, and deploy across cloud providers and on-premises systems. For more information about Tekton, see the Tekton documentation.

Overview of Tekton

Tekton is a cloud-native, open-source framework for creating CI/CD systems. It allows developers to build, test, and deploy across cloud providers and on-premises systems. For more information about Tekton, see the Tekton documentation.

Install Tekton

This topic tells you how to install Tekton Pipelines from the Tanzu Application Platform package repository.

Note

Follow the steps in this topic if you do not want to use a profile to install Tekton Pipelines. For more information about profiles, see Components and installation profiles.

Prerequisites

Before installing Tekton Pipelines, complete all prerequisites to install Tanzu Application Platform.

Install Tekton Pipelines

To install Tekton Pipelines:

1. See the Tekton Pipelines package versions available to install by running:

   ```
   tanzu package available list -n tap-install tekton.tanzu.vmware.com
   ```

   For example:

   ```
   $ tanzu package available list -n tap-install tekton.tanzu.vmware.com
   \ Retrieving package versions for tekton.tanzu.vmware.com...
     NAME                     VERSION  RELEASED-AT
     tekton.tanzu.vmware.com  0.30.0   2021-11-18 17:05:37Z
   ```

2. Install Tekton Pipelines by running:

   ```
   tanzu package install tekton-pipelines -n tap-install -p tekton.tanzu.vmware.co
   m -v VERSION
   ```

   Where `VERSION` is the desired version number. For example, `0.30.0`.

   For example:

   ```
   $ tanzu package install tekton-pipelines -n tap-install -p tekton.tanzu.vmware.
   com -v 0.30.0
   - Installing package 'tekton.tanzu.vmware.com'
   \ Getting package metadata for 'tekton.tanzu.vmware.com'
   / Creating service account 'tekton-pipelines-tap-install-sa'
   / Creating cluster admin role 'tekton-pipelines-tap-install-cluster-role'
   / Creating cluster role binding 'tekton-pipelines-tap-install-cluster-rolebindi
   ng'
   / Creating package resource
   - Waiting for 'PackageInstall' reconciliation for 'tekton-pipelines'
   - 'PackageInstall' resource install status: Reconciling


    Added installed package 'tekton-pipelines'
   ```

3. Verify that you installed the package by running:

   ```
   tanzu package installed get tekton-pipelines -n tap-install
   ```

   For example:

   ```
   $ tanzu package installed get tekton-pipelines -n tap-install
   \ Retrieving installation details for tekton...
   NAME:                    tekton-pipelines
   PACKAGE-NAME:            tekton.tanzu.vmware.com
   PACKAGE-VERSION:         0.30.0
   STATUS:                  Reconcile succeeded
   CONDITIONS:              [{ReconcileSucceeded True  }]
   USEFUL-ERROR-MESSAGE:
   ```

   Verify that `STATUS` is `Reconcile succeeded`.

Configure a namespace to use Tekton Pipelines

This section covers configuring a namespace to run Tekton Pipelines. If you rely on a SupplyChain to create Tekton PipelinesRuns in your cluster, skip this step because namespace configuration is covered in Set up developer namespaces to use your installed packages. Otherwise, perform the steps in this section for each namespace where you create Tekton Pipelines.

Service accounts that run Tekton workloads need access to the image pull secrets for the Tanzu package. This includes the `default` service account in a namespace, which is created automatically but is not associated with any image pull secrets. Without these credentials, PipelineRuns fail with a timeout and the pods report that they cannot pull images.

To configure a namespace to use Tekton Pipelines:

1. Create an image pull secret in the current namespace and fill it from the `tap-registry` secret. For more information, see Relocate images to a registry.
2. Create an empty secret, and annotate it as a target of the secretgen controller, by running:

   ```
   kubectl create secret generic pull-secret --from-literal=.dockerconfigjson={} --type=kubernetes.io/dockerconfigjson
   kubectl annotate secret pull-secret secretgen.carvel.dev/image-pull-secret=""
   ```

3. After you create a `pull-secret` secret in the same namespace as the service account, add the secret to the service account by running:

   ```
   kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "pull-secret"}]}'
   ```

4. Verify that a service account is correctly configured by running:

   ```
   kubectl describe serviceaccount default
   ```

   For example:

   ```
   kubectl describe sa default
   Name:                default
   Namespace:           default
   Labels:              <none>
   Annotations:         <none>
   Image pull secrets:  pull-secret
   Mountable secrets:   default-token-xh6p4
   Tokens:              default-token-xh6p4
   Events:              <none>
   ```

   Note

   The service account has access to the `pull-secret` image pull secret.

For more details about Tekton Pipelines, see the Tekton documentation and the GitHub repository.

For information about getting started with Tekton, see the Tekton tutorial in GitHub and the getting started guide in the Tekton documentation.

Note

Windows workloads are deactivated and cause an error if any Tasks try to use Windows scripts.